TREBALL DE FI DE CARRERA

TÍTOL DEL TFC: Algorismes Multicast en Xarxes Overlay

TITULACIÓ: Enginyeria Tècnica de Telecomunicació, especialitat Telemàtica

AUTOR: Víctor A. van Creij

DIRECTOR: Javier Ozón Górriz

DATA: 8 d'Octubre de 2009

Títol: Algorismes Multicast en Xarxes Overlay

Autor: Víctor A. Van Creij

Director: Javier Ozón Górriz

Data: 8 d'Octubre de 2009

Resum

En aquest treball (continuació d'altres TFCs), es proposa l'estudi i desenvolupament d'una família d'algorismes per l'encaminament multicast, on un usuari anomenat arrel o font ha d'enviar informació a un grup determinat de nodes. Tot i que aquest grup d'algorismes pot implementar-se en qualsevol nivell de la torre de protocols, en aquest document es planteja el seu estudi al nivell d'aplicació. Aquests algorismes busquen la creació de taules d'encaminament òptimes entre els diferents usuaris, de tal manera que es pugui minimitzar el temps en què la informació arriba a tots els usuaris.

Els algorismes aprofiten el retard de la transmissió d'un missatge per reenviar el paquet cap a un tercer node. Prenent com a base un algorisme dissenyat per a la transmissió d'un únic missatge, s'han creat algorismes adaptats a l'original, els quals permeten l'enviament de més d'un missatge d'informació i tracten a més d'evitar problemes de congestió. Per aquest motiu, s'ha limitat inicialment el nombre de vegades que el node font pot enviar la informació i s'ha definit a més un temps de cadència màxim per als diferents nodes de la xarxa. De la mateixa manera, s'han definit mecanismes amb la intenció d'evitar l'aïllament de nodes dins del grup multicast.

Per tal d'estudiar el comportament dels algorismes, s'ha dissenyat una xarxa virtual que modelitza la xarxa IP, a la qual hem afegit un nombre determinat d'usuaris (que conformen finalment el grup multicast), tot amb determinades característiques d'ample de banda i retard. Posteriorment, s'ha definit sobre aquesta xarxa una segona xarxa overlay formada pels nodes d'usuari, definida sobre el nivell d'aplicació. És en aquesta segona xarxa on s'apliquen els algorismes d'encaminament.

Finalment, s'han comparat els diferents algorismes proposats (alguns recollits de treballs anteriors) per tal d'escollir el millor en cada cas. Com era previsible, els algorismes més ràpids poden presentar problemes de congestió. D'altra banda els algorismes definits per evitar la congestió presenten un retard molt més elevat. D'aquesta manera, al final del treball proposem l'aplicació dels algorismes més ràpids, tot limitant la cadència del node font a la cadència més elevada dels nodes d'usuari. D'aquesta forma aprofitem d'una banda la velocitat dels arbres d'encaminament més ràpids, i d'altra evitem la possibilitat de tenir congestió en cap dels nodes de la xarxa.

Title: Multicast Algorithms for Overlay Networks

Author: Víctor A. Van Creij

Director: Javier Ozón Górriz

Date: October, 8th 2009

Overview

This work (which follows previous studies) proposes the development of a family of algorithms that solves the multicast routing problem, where a given node called root has to send information to a certain group of receiving nodes. Although this group of algorithms can be applied at any level of the protocol stack, this paper studies its performance at application level. This family of algorithms searches optimal routing tables between different user nodes in such a way that the total time needed to reach all users is minimised.

The algorithms take advantage of the delay in transmission time of a message between one peer and another, to forward the data packet to a third peer. Based on the algorithm designed to send a single packet, adapted algorithms have been created which allow sending more than one message, and besides avoid congestion problems. With this purpose, the number of times that the root can send a packet has been initially restricted, and a maximum cadence time has been defined for the different nodes of the network. Moreover, we have applied mechanisms to guarantee full connectivity within the multicast group.

With the goal to study the behaviour of the different algorithms, a virtual network has been designed, which models the IP network. We have also added to the former network a certain number of users (which finally form the multicast group), all with certain characteristics of bandwidth and delay. Subsequently, we have drawn on top of the first network an overlay network, defined at the application level. It is in this second network where the routing algorithms have been developed.

Finally, we compare the results of the different algorithm proposals (some of them taken from previous works), aiming to select the best in each case. As expected, the fastest algorithms could cause congestion problems. On the other hand, the algorithms defined to avoid congestion, result in increased delay. Therefore, the final proposal of this paper is to employ the fastest algorithms, limiting the root cadence time to the highest user node cadence. In this way, on one hand we take advantage of the speed of the fastest rooting trees, and on the other hand we avoid congestion all over the network.

ÍNDEX

INTRODUCCIÓ

En el transcurs dels últims anys, el nombre d'ordinadors connectats a Internet ha crescut considerablement, com també el nombre d'aplicacions que poden ser executades sobre ells. Molt sovint, aquestes aplicacions consisteixen en transmissions entre un ordinador i un altre, o entre un ordinador i un grup d'aquests. Des del començament de les xarxes d'ordinadors, les transmissions unicast (comunicacions un contra un) i broadcast (comunicacions entre una màquina i un grup indefinit d'aquestes, és a dir, el que es coneix com a un contra tots) estan disponibles per a ésser utilitzades; ara bé, les transmissions des d'un únic ordinador fins a un grup definit de receptors és un problema que encara no ha estat resolt de forma satisfactòria. Moltes aplicacions poden utilitzar aquests tipus de transmissions: videoconferències, jocs multijugador en línia o l'intercanvi d'arxius dins una xarxa P2P.

Les adreces multicast sobre IP són una possible solució per aquest problema, però també presenten alguns inconvenients rellevants. Primer, el nombre d'adreces multicast és baix, la qual cosa implica un nombre limitat i reduït de grups. I segon, alguns dels algorismes i protocols d'encaminament multicast són complexes i, el que és més important, tots els equips de xarxa (com els routers i els switches) han de saber interpretar aquests protocols. En altres paraules, s'hauria de canviar bona part de l'equipament de les xarxes IP per poder proporcionar arreu aquest servei multicast.

La solució que es presenta en aquest treball (continuació d'altres [7,17,18]) proposa una altra estratègia pel problema de l'encaminament multicast. Es presenta una família d'algorismes que poden ser utilitzats en qualsevol nivell (xarxa, enllaç, aplicació) gràcies al seu grau d'abstracció. Aquests algorismes presenten les següents característiques principals:

- Implementació simple.
- Retard reduït de transmissió.
- Alta escalabilitat.

Aquests algorismes es basen en una premissa molt simple: des d'un node origen (conegut també com a arrel/*root*), i tenint un grup de nodes destí (que formen el grup multicast), l'algorisme tracta de trobar el camí òptim que minimitza el retard total de la transmissió quan s'envien dades des del node origen fins als destinataris del grup.

Al començament [7,17,18], l'algorisme es va definir per tal d'enviar un únic paquet en una xarxa molt homogènia. Després d'això, s'han proposat escenaris i xarxes més complexes, i s'han afegit a més diferents condicions de comportament per tal de millorar l'algorisme inicial, d'acord amb les característiques de les xarxes reals.

Pel seu estudi i comprovació, s'han aplicat els algorismes en diferents xarxes simulades. En particular, hem treballat amb una representació virtual de la xarxa d'Internet, la qual consta de xarxes de trànsit (amb velocitats i retards

elevats) i xarxes d'accés (amb velocitats i retards petits, i connectades als nodes de trànsit).

A continuació, hem definit un conjunt de xarxes overlay, tot afegint a les xarxes anteriors un seguit de nodes d'usuari (o peers) connectats a través de les xarxes d'accés. L'objectiu d'aquesta operació és la simulació d'un escenari real d'Internet: per exemple, un grup d'amics que es connecten a Internet utilitzant les seves respectives connexions ADSL amb diferents operadors, i volen jugar un joc en línia sense utilitzar cap servidor central. Aleshores, el grup d'amics definirà un grup multicast amb l'objectiu de transmetre's les dades de la forma més eficient possible. De totes maneres, aquest és un simple exemple, ja que els algoritmes proposats en el present projecte poden ser utilitzats en diferents escenaris, com una xarxa Ethernet i també en el nivell d'aplicació.

Un cop hem definit la xarxa overlay amb els nodes que formen el grup multicast, hem aplicat els diferents algorismes d'encaminament. La nostra execució retorna informació sobre l'arbre d'encaminament, la cadència de cada node, l'instant de temps en que qualsevol node rep el paquet i el retard total de la transmissió. També s'han comparat els resultats dels diferents algorismes per tal d'escollir el millor en cada cas. Com s'esperava, els algorismes més ràpids presenten sovint inconvenients de congestió, mentre que els algorismes definits per evitar la congestió presenten un retard més gran.

La memòria d'aquest treball es divideix de la següent manera: primer, s'explica el problema de la transmissió multicast i es presenta un possible model de xarxa d'Internet. La segona secció és una petita introducció a la teoria de grafs i a l'algorisme de Dijkstra, que és utilitzat més endavant en les simulacions, per obtenir el camí mínim entre tots els nodes de la xarxa. La tercera secció defineix els algorismes, començant pel model inicial, utilitzat per enviar únicament un paquet. La quarta presenta les aplicacions amb què hem generat les xarxes overlay, programat els algorismes i definit la lectura i processament de resultats. Finalment, la cinquena i última secció conté els paràmetres per generar les xarxes overlay, els resultats i les conclusions, a més de les línies futures de treball.

1. TRANSMISSIONS MULTICAST I MODELATGE DE LA XARXA IP

1.1 Transmissions multicast

En totes les xarxes de comunicació, incloent les xarxes de computadores, existeix algun tipus de topologia de transmissió. La més usual és la transmissió punt a punt, com la comunicació estàndard de telèfon (en què les dades, en aquest cas de veu, viatgen des d'un origen únic fins a un destí determinat), o la comunicació broadcast, que consisteix en la transmissió de dades des d'un origen únic fins a tots els receptors, sense especificar el seu nombre (com en la transmissió FM en ràdio). D'altra banda, quan es desitja transmetre dades des d'un punt fins a un subconjunt definit i conegut de receptors, l'assumpte es complica de forma considerable. Això es el que coneix com transmissió multicast, on un nombre definit de clients pot rebre el mateix flux d'una única font.

Les aplicacions multicast inclouen conferències de vídeo, jocs multijugador en línia, comunicacions corporatives, aprenentatge a distància, distribució de software, notícies, etc. Dins el context de les xarxes IP, la transmissió multicast va ser proposada per ésser implementada a la capa de xarxa [6], però s'ha utilitzat de manera generalitzada [2]. Multicast IP defineix un grup multicast on els clients reben el flux d'una única font, que envia els paquets al grup de destí a través dels routers multicast. D'aquesta forma, en enviar un únic paquet a la xarxa i tot deixant que la intel·ligència de la mateixa reprodueixi el paquet només quan sigui necessari, l'ample de banda i els recursos de la xarxa són explotats d'una manera eficient. Ara bé, existeixen alguns inconvenients en el sistema multicast IP, atès que en el disseny original d'IP no es va considerar la possibilitat de tenir comunicacions multicast, i es tracta per tant d'un afegit al protocol IPv4. En primer terme, el rang de les adreces IP per crear grups multicast és molt limitat i la majoria d'aquestes estan reservades. A més, es fan servir algorismes complexes (com és el cas de PIM) per realitzar l'encaminament multicast sobre la xarxa IP, i el que és més important, els elements de la xarxa (com per exemple els routers) han de saber interpretar aquests algorismes i els seus missatges per poder dur a terme la transmissió multicast.

Aquestes raons fan que la comunicació multicast en IPv4 sigui complexa. A més, en IPv4 el grup multicast es crea al nivell tres de la pila de protocols (nivell de xarxa), de forma que una aplicació que necessiti definir i manegar un grup multicast es trobarà amb serioses dificultats per administrar tot el grup *multicast*.

Les dificultats de la transmissió multicast sobre IP han permès el desenvolupament d'altres aproximacions a la capa d'aplicació, utilitzant arquitectures P2P [20]. En una aproximació multicast a la capa d'aplicació, coneguda com *overlay* multicast, els peers participants del grup s'autoorganitzen dins una topologia *overlay* per a la distribució de les dades. En

aquest tipus de topologia cada aresta correspon a un camí unicast entre dos nodes finals o peers. Totes les funcionalitats relacionades amb multicast són implementades pels peers en lloc dels routers amb l'objectiu de dibuixar una transmissió multicast eficient. Com és lògic, la capa d'aplicació multicast no és tan eficient com la capa de xarxa multicast ja que ofereix retards i consums d'ample de banda més elevats, així com una menor estabilitat de l'arbre multicast.

Les comunicacions de dades en temps real, com poden ser concerts o esdeveniments esportius, són especialment sensibles al retard, però també al *jitter* (variació del retard). En el cas dels jocs en línia, la informació de cada jugador ha d'ésser enviada a la resta de participants dins un determinat marge de temps per tal de preservar l'estat del joc. Pel cas de les aplicacions sota demanda de vídeo, en què el retard no és tan important, pren una rellevància especial el *jitter*, ja que en aquest context el que més importa és la cadència amb què els paquets arriben al destí, és a dir, la diferència entre els retards dels diferents paquets (variació fonamental per poder reconstruir el vídeo sense interrupcions). En aquest projecte es presenta un arbre multicast amb l'objecte de minimitzar el retard total de la transmissió, és a dir, el temps transcorregut entre l'instant en què és comença la transmissió del primer paquet i el moment en què les dades són rebudes per l'últim node del grup.

Existeixen nombrosos estudis i propostes per la capa d'aplicacions multicast. Aquests estudis es focalitzen sobretot en protocols per obtenir la màxima eficiència en la construcció i gestió de l'arbre multicast. N'hi ha dues aproximacions bàsiques per aquest problema: nodes fixes i nodes dinàmics basats en *overlay*. La primera proposta [12,20] situa estratègicament uns nodes especials dins de la xarxa que formen, quan alguna aplicació requereix el servei, la xarxa *overlay* multicast. Encara que l'arbre multicast és bastant estable i fàcil de mantenir, aquesta solució presenta inconvenients semblants al multicast sobre IP [2]. Pel cas dels nodes dinàmics, com els de [4,19,23], tot el grup de membres s'autoorganitza per formar l'arbre *overlay* multicast i s'encarrega de totes les funcions. En aquest context, en que és freqüent que s'afegeixin nodes o que n'abandonin el grup, l'adaptació de la xarxa pren un paper important i ha de ser presa en consideració per poder realitzar-se de forma eficient.

1.2 Modelatge de la xarxa IP

Com ja s'ha comentat, la xarxa *overlay* definida per les transmissions multicast s'ha de formar en la pràctica sobre la xarxa real. Aquesta "xarxa real", quan ens referim a xarxes de computadores, és Internet. Com la utilització de xarxes reals per a l'estudi de la congestió i l'encaminament sota diferents algorismes d'encaminament és d'una gran complexitat, en el nostre treball s'utilitzaran com a primera aproximació simulacions, tot assumint que el model utilitzat és una "bona" abstracció de la xarxa real.

Els models més habituals per les xarxes de computadors són:

- Topologies regulars, com anells, arbres, estrelles, etc.
- Topologies conegudes com l'ARPAnet o la connexió de xarxes NFSnet.
- Topologies generades aleatòriament.

Aquest tres casos presenten, com és obvi, certes limitacions: les topologies regulars o les topologies "conegudes" com l'ARPAnet, només representen una part de les xarxes antigues i actuals, mentre que les aleatòries no representen la xarxa real. Aquest extrem ha prendre's en consideració, ja que el comportament d'un algoritme pot variar considerablement d'una topologia a una altra.

Una altra forma de modelar una xarxa és la utilització d'un model jeràrquic. Existeixen dos models clàssics: *N-Level* i *Transit-Stub*. El primer comença amb un graf aleatori connex, i després substitueix els nodes recursivament per nous graf connexos. El segon model, *Transit-Stub*, s'explica a continuació.

1.3 El model de xarxa *Transit-Stub*

En el nostre cas, hem triat un modelatge en forma de graf. Per dur a terme aquest propòsit, s'ha pres com a punt de partida el model presentat per Ellen W. Zegura, Kenneth L. Calvert i Samrat Bhattacharjee [22]. En aquest model, els nodes representen els routers i switches, i les arestes els camins entre els elements interconnectats. El model no inclou usuaris individuals, és a dir, no es tenen en compte els terminals finals, sinó simplement l'estructura lògica de la xarxa.

La figura 1.1 presenta una xarxa creada segons aquest procediment. En primer lloc, es crea un graf connex aleatori, on cada node representa un domini de trànsit complet (dos dominis segons la figura). Aleshores, cada node és substituït per un nou graf connex, el qual representa la xarxa troncal per aquest domini de trànsit (amb cinc i tres nodes respectivament, a la figura). Després, per a cada node en cada domini de trànsit, es generen nous grafs aleatoris connexos, que representen el domini d'accés afegit a cada node (anomenats dominis *stub*). Finalment, afegim un determinat nombre d'arestes entre els nodes dels dominis de trànsit i els nodes dels dominis *stub* (o també entre nodes pertanyents a diferents dominis *stub*). Com tots els grafs generats són connexos, el graf resultant també és un graf connex. Qualsevol d'aquets grafs aleatoris pot ser creat amb un model senzill de grafs planars.

Els paràmetres necessaris per crear un model de xarxa utilitzant el model de *Transit-Stub* són:

- *T:* nombre de dominis de trànsit.
- N_t: nombre promig de nodes per domini de trànsit.
- K: nombre promig de dominis d'accés per domini de trànsit.
- N_s: nombre promig de nodes per domini de *stub*.

Amb aquests paràmetres, es poden utilitzar les eines descrites en la secció 4.1 per crear la connexió de xarxes (és a dir, la interconnexió a la capa de xarxa dels diferents dispositius). En tot cas, el model que obtenim no és suficient per fer el nostre estudi, ja que, com hem vist, no defineix els *hosts* individuals o, dit d'una altra manera, els equips d'usuari. Per tant, a més de tenir la xarxa troncal (amb nodes de trànsit *T*, i amb nodes d'accés *S*), necessitarem els nodes terminals o d'usuari, que actuaran com a membres del grup multicast entre els quals volem enviar la informació. Aquests nodes d'usuari s'han afegit mitjançant una aplicació en Java (descrita a la secció 4.2), que connecta de manera aleatòria cadascun dels nodes d'usuari del grup multicast amb un dels nodes *stub* del graf. En la pràctica, els nodes d'usuari representen ordinadors que, mitjançant una línia d'accés, es connecten punt a punt amb un node d'accés a Internet (un ISP, representat per un node stub *S*), per formar part d'aquesta manera de la xarxa *overlay*.

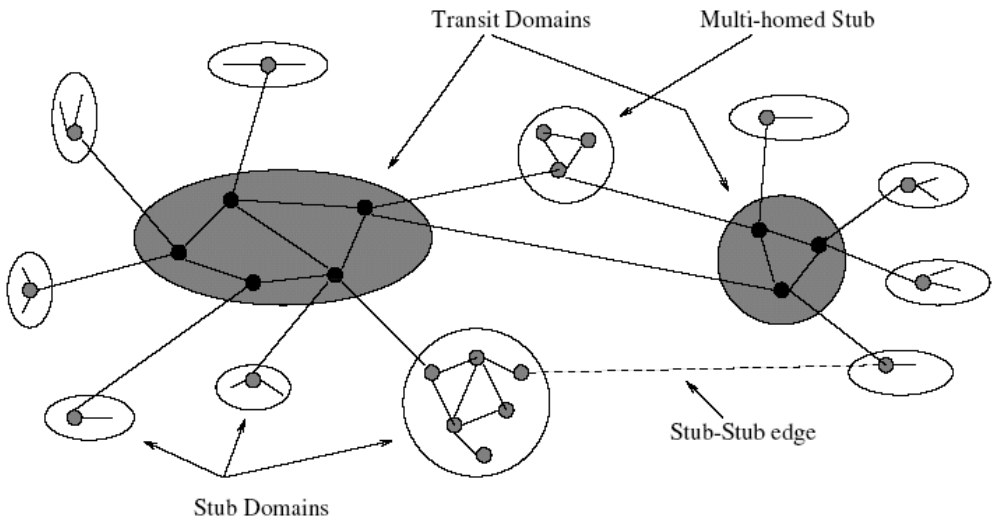


Fig.1.1 *Transit-Stub* model

2. TEORIA DE GRAFS

En aquesta secció presentem una breu introducció a la teoria de grafs i definim a més l'algorisme de Dijkstra que troba el camí més curt entre dos nodes qualssevol d'un graf. Amb aquestes eines, podrem modelar més endavant la xarxa d'Internet i crear els grafs *overlay* que conformin el grup multicast d'intercanvi d'informació.

2.1 Definicions

Un *graf G* és un parell (*V(G),E(G)*) en el qual *V(G)* és un nombre finit d'elements anomenats *nodes* o *vèrtexs*, i *E(G)* és un nombre finit de parells no ordenats de nodes anomenats *arestes* o *arcs*. L'ordre *n* d'un graf *G=(V,E)* és el nombre de nodes, o el que és equivalent, el cardinal de *V(G)*. Igualment, es defineix la mida *E* d'un graf *G=(V,E)* com el nombre d'arestes del graf, això és, el cardinal d'*E(G)*. Normalment, un graf es representat gràficament amb punts com a nodes, i línies que uneixen aquets punts com a arestes, tal i com es pot veure a les figures 2.1 i següents. Quan el graf *G* no conté cap aresta repetida ni cap node adjacent a si mateix, es diu que és un *graf simple*.

Tant els nodes com les arestes poden tenir funcions d'etiquetatge, això és, una aplicació $\Phi:V(G) \rightarrow Z$ o $\Phi':E(G) \rightarrow Z$ de forma que cada node i/o aresta té un nombre enter associat (més genèricament, el destí de les aplicacions pot ser qualsevol conjunt, com per exemple els nombres reals). Aquestes etiquetes, també conegudes com a *pesos*, poden ser utilitzades per identificar els elements del graf, o per establir algunes propietats d'aquets elements, com l'ample de banda de l'enllaç o el tipus de node.

Es diu que dos nodes *u* i *v* són adjacents (o veïns) quan estan connectats per una aresta *uv*. En aquest cas, els nodes *u* i *v* són adjacents a l'aresta *uv*, i l'aresta *uv* es diu també que és adjacent als nodes *u* i *v*. Dues arestes són adjacents quan tenen un node en comú. El grau $\delta(v)$ d'un node *v* és el nombre d'arestes adjacents a *v*.

Una seqüència d'arestes és una successió d'arestes consecutives v_0v_1, v_1v_2, v_2v_3, .., $v_{m-1}v_m$. Aquesta seqüència representa un camí continu per al graf. Una seqüència d'arestes no repetides s'anomena *camí*, i si no existeixen nodes repetits, s'anomena *camí simple*. Un *cicle* és un camí que comença en un node i finalitza en aquell mateix node.

Dos nodes *u* i *v* estan connectats si *G* conté un camí des d'*u* fins a *v*. Si tots els parells de nodes de *G* estan connectats, llavors el graf *G* és un *graf connex*. Un *component connex* és tot subgraf connex màxim de *G*. Cada node de *G* pertany a un únic component connex, i també cada aresta. La distància *d(u,v)* entre dos nodes *u* i *v* d'un graf *G* és la longitud del camí més curt entre ells, és a dir, el nombre mínim d'arestes que s'han de travessar per anar des d'un node fins a l'altre. Si el graf no és connex i existeixen dos nodes associats a dos components connexos diferents, aleshores es diu que la seva distància és

infinita. L'excentricitat $\varepsilon(v)$ d'un node en un graf G és la màxima distancia des de v fins a un altre node qualsevol del graf. El diàmetre $D(G)$ d'un graf G és la màxima excentricitat entre tots els node del graf, i el radi $R(G)$, la mínima.

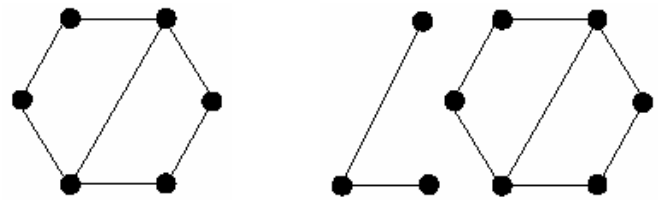


Fig. 2.1 Exemple d'un graf connex (esquerra) i d'un graf no connex amb dos components connexos (dreta)

Un arbre és un graf connex sense cicles, o dit d'una altra manera, un graf en què qualsevol parell de nodes està connectat per un únic camí. Un graf K_n *complet* és un graf simple tal que tots els nodes són adjacents entre si, i per tant la seva mida és $n(n-1)/2$. Si tots els nodes del graf tenen el mateix grau, aleshores es diu que el graf és *regular* i, en particular, si tots els nodes del graf tenen grau r, llavors es diu que és un graf regular de grau r, o un graf *r-regular*.

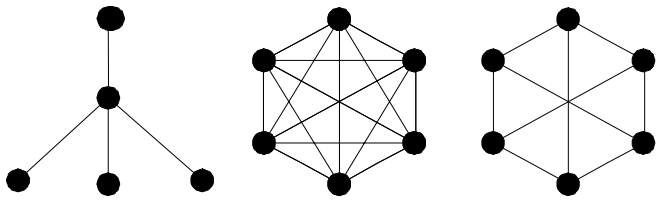


Fig. 2.2 Exemples d'un arbre, un graf complet i un graf regular

Un *graf directe o dígraf* G és un parell ordenat $(V(G),A(G))$, on $V(G)$ és un conjunt finit de nodes, i $A(G)$ és un conjunt finit de parells ordenat de nodes, anomenats arestes. Una aresta $e=vw$ es considera que es dirigeix des del node v fins al node w; w és anomenat el cap i v la cua de l'aresta. És important remarcar que els arcs vw i wv són diferents, i l'aresta wv és el que s'anomena aresta invertida de vw. Si G no té cap aresta des d'un node fins a ell mateix (anomenada volta), i totes les arestes de G són diferents, aleshores es diu que G és un *dígraf simple*. Si G és un dígraf, el graf que s'obté eliminant les direccions de les arestes s'anomena *graf base* de G.

Totes les definicions descrites per un graf simple poden ésser esteses a un dígraf. Es poden definir aplicacions d'etiquetatge $\Phi:V(G) \rightarrow Z$ o $\Phi':E(G) \rightarrow Z$ sobre els node i les arestes, i també seqüències finites d'arestes v_0v_1, v_1v_2, v_2v_3, .., $v_{m-1}v$, anomenades *camins,* on ni els arcs ni els node es repeteixen.

Es diu que un dígraf G és dèbilment connex si el graf base de G és un graf connex. D'altra banda, G és fortament connex si conté un camí com a mínim per a tots els parells de nodes $v,w \in V(G)$. Mentre que tots els dígrafs fortament

connexos són dígrafs dèbilment connexos, no tots els dígrafs dèbilment connexos són fortament connexos.

2.2 El problema del camí més curt. L'algorisme de Dijkstra

En aquest treball es vol estudiar l'aplicació d'alguns algorismes per tal d'optimitzar la transmissió d'informació sobre un determinat grup de nodes. Més tard es podrà veure com treballen aquests algorismes, però per ara es pot avançar que aquestes operacions estan basades en el fet que les computadores dins el mateix grup poden reenviar la informació.

Per dur a terme aquesta tasca, primer necessitarem trobar els nodes més propers a un usuari en particular, i el cost o retard d'enviar les dades. Aquesta qüestió, que planteja el problema del camí més curt entre dos nodes qualssevol, es resol amb un algorisme simple anomenat Dijkstra. En concret, l'algorisme de Dijkstra troba el camí més curt entre un node *u* del graf i la resta de nodes. La longitud o distància d'un camí es mesura com la suma dels pesos o les etiquetes de les arestes que formen el camí. En aquest algorisme cada node *v* de *G(V,E)* té associada una etiqueta *L(v)*. Aquesta etiqueta mostra la longitud del camí més curt conegut per anar des d'un node fixat *u* fins a un altre *v*. Al començament, el valor de *L(v)* és el valor del pes *w(u,v)* de l'aresta que connecta els nodes *u* i *v*, i, si l'aresta no existeix, aquest valor és infinit. Igualment *L(u)=0,* de tal forma que es considera que el cost de quedar-se en el mateix node és 0. L'algorisme treballa amb un conjunt de nodes $T \subset V$, que augmenta conforme s'executa l'algorisme i que conté els nodes pels quals ja s'ha trobat el camí més curt (sempre des d'*u* fins a ells). Al final de l'execució de l'algorisme, *L(v)* conté el cost (retard o distància) del camí més curt per anar des del node *u* fins a qualsevol node *v* de *V(G)*.

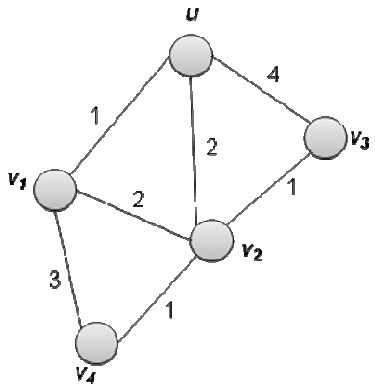
1. **per a tot** $v \neq u$ $L(v) = w(u,v)$
2. $L(u) = 0$
3. $T = \{u\}$
4. **mentre** $T \neq V$
   Inici
5. troba $v' \notin T$ tal que $\forall\ v \notin T$ $L(v') \leq L(v)$
6. $T = T \cup \{v'\}$
7. **per a tot** $v \notin T$ tal que v' és adjacent a v
   si $L(v) > L(v') + w(v',v)$
   llavors $L(v) = L(v') + w(v',v)$ **fi si**

   fi per a
   fi mentre

Fig. 2.3 Algorisme de Dijkstra

A cada iteració, l'algorisme afegeix un nou node a la seva llista *T*. Això es fa escollint un node *v'*, que no pertany a la llista *T* i que posseeix el menor etiquetatge *L(v')*. En altres paraules, l'algorisme escull el node *v'* que encara no pertany a la llista *T* i que dóna el camí més curt des d'*u* a *v'*. Un cop fet això, els

nodes connectats directament a *v'* han d'actualitzar les seves etiquetes (ja que ara podrem traçar un nou camí des d'*u* fins a aquests nodes passant per *v'*). Finalment, el node *v'* es afegit a la llista *T*. Aquest procés es repeteix fins que tots els nodes del graf són afegits a la llista. La figura 2.3 mostra un esquema més detallat de l'algorisme i la 2.4 una execució per a un graf particular.



Iteration	V'	L(u)	$L(v_1)$	$L(v_2)$	$L(v_3)$	$L(v_4)$	T
0	-	0	1	2	4	∞	{u}
1	v_1	0	1	2	4	4	{u, v_1}
2	v_2	0	1	2	3	3	{u, v_1, v_2}
3	v_3	0	1	2	3	3	{u, v_1, v_2, v_3}
4	v_4	0	1	2	3	3	V

Fig. 2.4 Exemple de l'execució de Dijkstra per a un node *u*

Es pot demostrar que aquest algorisme és òptim; per fer-ho, provarem que cada vegada que un node *v'* s'afegeix a *T*, l'etiqueta *L(v')* indica la distància mínima de *u* a *v'*. En efecte, suposem que això no sigui així, i que existeix un camí més curt de *u* a *v'*. Sigui w_2 el primer node adjacent a *u* pel qual passa aquest nou camí de *u* a *v'* i de distància menor a *L(v')*. Aquest node w_2, per construcció, haurà de pertànyer a *T*, ja que la distància de *u* a w_2, que l'algorisme coneix des de la primera iteració, ha de ser menor que *L(v')*. El mateix argument es pot repetir pel següent node del camí, w_3, que tindrà una distància des d'*u* més petita que *L(v')* calculada en afegir-se w_2 a *T* i que per tant s'haurà afegit igualment a *T* , i així successivament fins a arribar a *v'*. D'aquí es dedueix que, si existís un camí més curt de *u* a *v'* que el que indica l'etiqueta *L(v')*, l'algorisme hauria anat afegint en *T* els nodes w_2,w_3,w_4... que formen aquest camí més curt, i per tant l'hauria trobat. Per últim, com que en acabar l'algorisme tots els nodes es troben a la llista *T*, l'algorisme de Dijkstra troba el camí més curt des d'un node *u* fins a qualsevol altre node del graf.

També és fàcil provar que aquest algorisme requereix un temps de càlcul d'ordre $O(n^2)$, cosa que, a la pràctica, significa que es poden trobar els camins òptims en temps de computació reduïts. Per determinar el mínim *L(v')* (línia 5 del pseudocodi de l'algorisme) necessitem *O(n)* comparacions, i la línia 7 no

necessita més de *n* assignacions. Aquestes dues línies es troben a la sentència *mentre* de la línia 4, que s'executa *(n-1)* vegades. Per tant, l'algorisme complet es pot executar en un temps d'ordre $O(n^2)$.

Per acabar, hem d'afegir que aquest algorisme, a més de calcular la longitud o el cost del camí més curt entre dos nodes, permet conèixer quina és la ruta que cal seguir per anar per aquest camí. Això s'aconsegueix afegint una nova etiqueta a cada node, de manera que cada vegada que s'actualitzi el valor de *L(v')* d'un determinat node *v'* s'afegirà el node *v* a partir del qual s'ha calculat el seu valor *L(v')*. D'aquesta manera, l'algorisme de Dijkstra no ens donarà només el cost de la ruta mínima per anar entre dos nodes, sinó que també obtindrem la successió de nodes que conduiran de manera òptima des de *u* fins a qualsevol node del graf.

3. ALGORISME D'ENCAMINAMENT MULTICAST

En aquesta secció, presentem el model del carter o *postal model* sobre el qual més endavant definirem la transmissió multicast en què, com s'ha dit, un node envia informació a un conjunt de nodes, anomenats en aquest context *peers*.

3.1 El model del carter

Per tal de millorar la transmissió de dades entre peers, A. Bar-Noy, i S. Kipnis introdueixen en [3] el *MPS(n) Postal Model*, que caracteritza un sistema *message-passing* o de missatge passarel·la, en el qual s'utilitzen tècniques de commutació de paquets i es defineix un paràmetre de latència $\lambda \geq 1$ (corresponent al temps que transcorre des que un node comença a enviar el missatge fins que aquest es completament rebut per un altre peer, és a dir, al temps de transmissió més el temps de propagació de l'enllaç). Aquest model, que nosaltres hem traduït com a *model del carter*, és utilitzat pels autors per estudiar l'impacte de les latències de les comunicacions en els algorismes de broadcast, en què un node envia informació a la resta de peers, i per xarxes completament connectades (és a dir, en les que tots els peers són adjacents a tots els peers, que és el que hem anomenat graf complet en el capítol anterior).

Al model del carter s'utilitza la latència de la informació (no confondre amb el paràmetre λ definit al paràgraf anterior; aquí ens referim al temps o "buit" que transcorre entre l'instant en què l'origen acaba d'enviar les dades i el destinatari les rep) per tal de millorar la taxa de la transmissió: en comptes de romandre en silenci esperant que la informació arribi al destí, el node font reenvia el paquet a altres nodes per tal d'inundar abans la xarxa. Igualment, els peers que ja han rebut un paquet poden retransmetre'l a altres nodes. Aquest model busca arbres d'encaminament òptims basats en els arbres de *Fibonacci*, en oposició als tradicionals arbres binomials. Encara que el model original presentat a [3] està pensat per transmissions broadcast, pot ésser estès d'una manera senzilla a les transmissions multicast, creant un conjunt de nodes de destí i transmetent en broadcast la informació sobre ells.

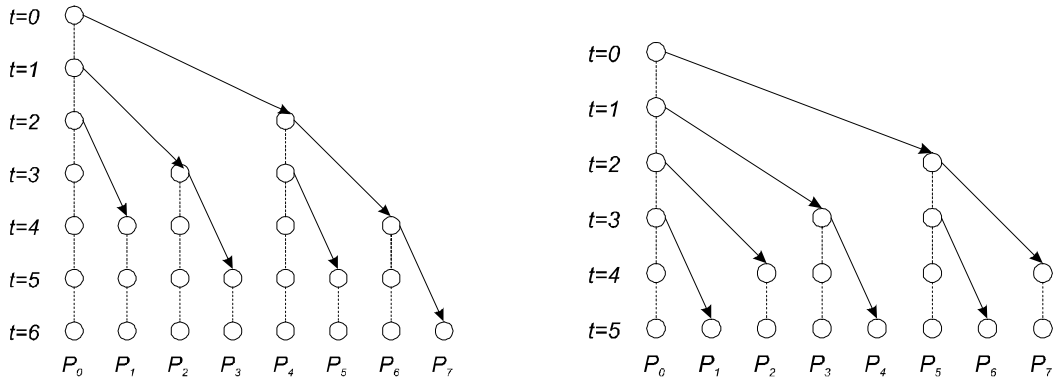


Fig. 3.1 Arbres de transmissió (Binomial a la l'esquerre i Fibonacci a la dreta)

A la figura 3.1 es pot veure una representació de la transmissió des de P_0 a set peers diferents utilitzant els arbres binomial i el de Fibonacci. En tots dos casos, cada node pot enviar un paquet en cada unitat de temps *t*, i el valor de la latència es de $\lambda=2$ (es necessiten dos unitats de temps per anar d'un node a un altre i per tant el temps de propagació és igual a una unitat de temps). L'arbre de l'esquerra utilitza un arbre binomial, i necessita 6 unitats de temps per enviar les dades a tots els peers. Contràriament, el de la dreta utilitza l'arbre de Fibonacci del model del carter, i només necessita 5 unitats de temps per cobrir tots els peers.

3.2. El model del carter estès

En aquets projecte, com a continuació d'altres [7,17,18], es treballa amb un model estès del model del carter que hem anomenat segons l'acrònim anglès *EMPS*. Es defineix un sistema de missatge passarel·la amb *n* peers, *EMPS(n,λ,μ)* amb una connectivitat *full-duplex* entre els peers $\{p_0,p_1,...p_{n-1}\}$ en el qual cada peer *p* pot enviar simultàniament un missatge a un peer *q* i rebre un altre missatge d'un altre peer *r*, d'acord amb els següents paràmetres:

- Per a cada peer *p* en un sistema de missatge passarel·la, es defineix un temps de transmissió μ_p com el temps que necessita *p* per transmetre un missatge *M* de longitud *l*. Es denota μ com el vector de tots els μ_p. En un context més detallat podem definir un temps de transmissió μ_{pq} per a cada parell *p* i *q* de nodes. En aquest cas μ és una matriu quadrada amb tots els temps de transmissió μ_{pq}.
- Per a cada parella de peers *p* i *q* en un sistema de missatge passarel·la, es defineix la latència de comunicació λ_{pq} entre dos peers *p* i *q*. Si en un temps *t* el peer *p* comença a enviar un missatge *M* al peer *q*, aleshores el peer *p* enviarà el missatge *M* durant l'interval de temps [t, t+ μ_p], i el peer *q* rebrà el missatge *M* durant l'interval [t+ λ_{pq} - μ_p, *t*+ λ_{pq}] com es mostra a la figura 3.2. Tal com s'ha definit, λ_{pq} és la suma del temps de transmissió μ_p del peer *p*, i el retard de propagació entre *p* i *q*. Es denota amb λ la matriu quadrada de totes les λ_{pq}.

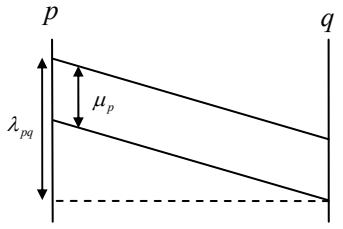


Fig. 3.2 The Extended Postal Model

Encara que una xarxa *overlay* normalment és una xarxa completament connexa (tots els peers estan directament connectats entre si) la representació del *EMPS(n,λ μ)* no requereix aquesta connectivitat total (és a dir, la xarxa no ha de ser un graf complet). També s'assumeix que el temps de processat dels

peers és negligible. Tanmateix, el model pot ésser fàcilment modificat quan un peer *p* té un temps de procés diferent de 0, afegint un temps de processat a la latència cada vegada que *p* retransmet el paquet o missatge per primer cop.

El model *EMPS(n,λ,μ)* és una generalització del *MPS(n)* de [3]. En aquest darrer, μ_p és la unitat per a tots els peers, λ_{pq} també és la mateixa per a qualsevol parella de nodes *p* i *q*; i finalment, la xarxa *overlay* és un graf complet. El model *EMPS(n,λ,μ)*, contràriament, considera una heterogeneïtat entre els peers, de tal forma que podem tenir diferents temps de transmissió per peers diferents i també diferents retards de propagació per a cada parella de peers.

Per simplicitat, s'assumeix que per als missatges de la mateixa mida la latència de comunicació és una constant en funció de temps. És a dir, no es considera la possible variació de la latència de la comunicació degut a la càrrega i els enllaços perduts de la xarxa física. Les xarxes a nivell d'aplicació utilitzen els serveis que proveeixen les capes inferiors de xarxa, com els protocols TCP/IP, per tal d'establir les comunicacions unicast *full-duplex* entre cada parell de peers. El terme de missatge es refereix a la peça de les dades enviada per un peer a un altre utilitzant els protocols de les capes inferiors.

Així es denota com *EMPS(n,λ,μ)* el sistema de missatge passarel·la amb *n* peers, una matriu de latència de comunicació *λ* i un vector (o matriu) de transmissió *μ*.

3.3 Algorisme *Single Message Multicast*

El problema de la transmissió multicast d'un missatge en un sistema de missatge passarel·la es defineix a continuació. Sigui p_0 un peer del model *EMPS(n,λ,μ)* que té un missatge *M* per enviar (en multicast) a un conjunt de peers receptors $R=\{p_1,p_2,\ldots,p_{n-1}\}$ en l'instant $t=0$. Hem de trobar un algorisme que minimitzi el temps de multicast t_M, és a dir, l'instant de temps en què l'últim peer de *R* rep el missatge *M*. Encara que el resultat de *EMPS(n,λ,μ)* és un arbre generador multicast (això és, un arbre que connecta tots els peers de la xarxa), la figura 3.3 mostra que aquest problema és diferent del conegut problema de l'arbre generador mínim. En el nostre problema, el retard de propagació entre dos peers *p* i *q* no és sempre el pes λ_{pq} de l'aresta que els uneix, atès que si el peer *p* ha reenviat el missatge a un altre peer abans de reenviar-l'hi a *q*, aleshores s'ha d'afegir a aquets retard el temps de transmissió μ_p.

A [3] els autors defineixen l'algorisme BCAST que proporciona el temps òptim pel cas dels arbres multicast amb connexió total, $\mu_p=1$ i $\lambda_{pq}=1$ per a qualsevol parella de peers (p,q). Aquest temps òptim està basat en una generalització dels nombres de Fibonacci, i els seus arbres són una generalització dels arbres de Fibonacci. Els autors també exposen a [3], que per qualsevol estratègia òptima, cada peer, un cop ha rebut el missatge *M*, l'ha de reenviar a un nou peer per cada unitat de temps. Aquesta idea també s'ha aplicat al nostre model *EMPS (n,λ,μ)* amb la diferència que ara les retransmissions dels missatges d'un

peer *p* han de tenir lloc cada temps de transmissió μ_p. L'algorisme que es proposa, anomenat SMM *Single Message Multicast*, es representat a la figura 3.4.

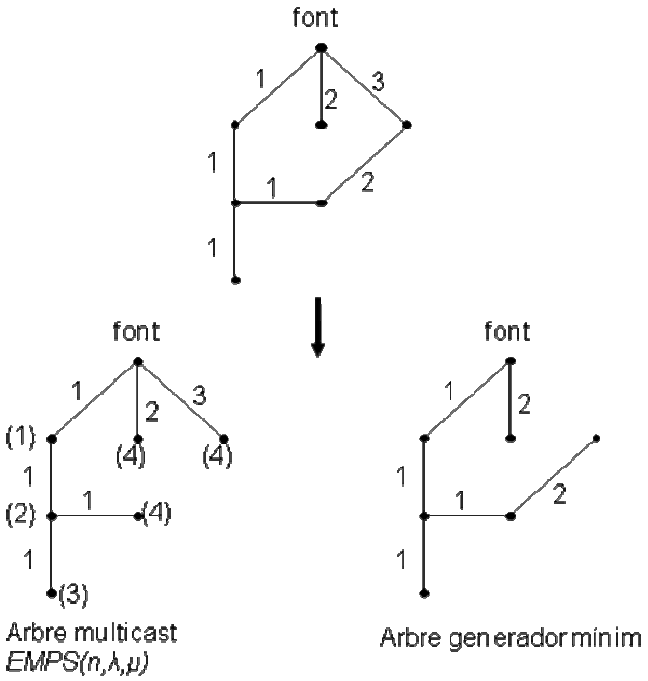


Fig. 3.3 Comparació entre *EMPS(n,λ,μ)* i l'arbre generador mínim. En aquest cas μ=1 per a tots els nodes i la latència és el pes de l'aresta. Entre parèntesi s'ha escrit el temps en què el missatge arriba a cada node

```
Data: EMPS(n,λ,μ)
Result: routing[i].send[j]
send ← 1;
routing[i].send[j] ← 0  ∀i, j ;
routing[i].tnext ← ∞  ∀i ;
routing[root].tnext ← lowest latency of root;
while send < n do
    i ← imin();
    next ← routing[i].index;
    routing[i].send[next] ← 1;
    update_i(routing[i].index);
    update_t(routing[i].tnext);
    update_i(routing[next].index);
    update_tn(routing[next].tnext);
    send ← send+1;
end
```

Fig. 3.4 Algorisme SMM

Les variables, les matrius i les funcions que utilitza l'algorisme són les següents:

- *routing[i].send[j]*: és la taula d'encaminament. Inicialment, tots els seus valors són 0. Quan acaba l'SMM, *routing[i].send[j]* igual a 1 significa que el peer *i* ha de reenviar el missatge al peer *j*. Si és igual a 0, aleshores el peer *i* no envia el missatge al peer *j*. Com cada peer té una llista ordenada dels seus veïns d'acord amb la seva latència (i.e. temps de transmissió més temps de propagació), un cop el peer *i* a rebut el missatge, el reenviarà al primer peer *j* tal que *routing[i].send[j]=1*. Després reenviarà el missatge al següent peer *k* amb *routing[i].send[k]=1* i així successivament.
- *i*: el peer que envia el missatge a cada pas.
- *next*: el peer que rep el missatge a cada pas.
- *routing[i].index*: apunta al peer més proper de *i* que encara no ha rebut el missatge. Entenem per peer més proper el que té una latència (temps de transmissió més temps de propagació) més petita respecte *i*.
- *routing[i].tnext*: el temps en el qual si un peer *i* envia el missatge, aquest arribarà al peer més proper dels peers no visitats. Aquest peer receptor és el *routing[i].index*.
- *imin()*: escull el peer amb *routing[i].tnext* més baix.
- *update_i()*: busca el peer més proper a *i* del conjunt de peers que encara no han rebut el missatge.
- *update_t()*: un cop el peer *i* a reenviat el missatge, *update_t()* calcula el següent valor per *routing[i].tnext*. Això és, sostreu del valor previ la última latència, i l'afegeix la següent latència (del peer més proper que encara no ha rebut el missatge) més el seu temps de transmissió.
- *update_tn()*: com *update_t()* però aplicat al peer que acaba de rebre el missatge. Al temps en què el node rep el missatge, s'afegeix la latència del peer més proper, escollit entre els que encara no han rebut el missatge.

El *modus operandi* de l'algorisme és senzill. A cada iteració SMM escull el peer que encara no ha rebut el missatge i que té el cost més baix, això és, el peer que encara no ha estat visitat i que pot rebre abans el missatge des de qualsevol dels peers que ja l'han rebut. Un cop el missatge ha estat rebut pel nou node, l'algorisme recalcula els temps d'arribada pels peers restants (considerant que el nou peer pot reenviar el missatge de manera immediata), escull el següent peer amb el temps d'arribada més baix i reenvia el missatge cap a ell. El càlcul dels temps d'arribada es fa sota l'assumpció que quan un peer ha acabat la transmissió d'un missatge a un altre peer, comença immediatament amb la retransmissió del missatge cap a un altre node destí. L'algorisme SMM és molt similar a l'algorisme de Dijkstra [8] amb la diferència que en l'*EMPS(n,λ,μ)* el retard de temps entre dos nodes *p* i *q* no és constant. De fet, en l' *EMPS(n,λ,μ)* el retard entre els peers *p* i *q* és igual a la suma de λ_{pq} i de μ_p multiplicat pel nombre de retransmissions prèvies del peer *p* (o, quan el temps de transmissió de *p* és variable segons el destinatari, a la suma de λ_{pq} i el temps total de totes les transmissions anteriors de *p*).

Considerem la xarxa representada a la figura 3.5, on els pesos de les arestes corresponen al retard de comunicació λ_{pq} (temps de transmissió més temps de propagació) entre els nodes de l'aresta. Per simplicitat, suposem que el temps de transmissió de tots els nodes és igual a la unitat. També assumim que el node origen del missatge *M* és el node p_0. En l'instant t=0, p_0 envia al peer p_1 el missatge, que arriba en t=10. En t=1, p_0 té el seu enllaç disponible i pot enviar el missatge *M* al següent node més proper p_2, el qual rebrà el missatge a t=11. Tanmateix, per als peers p_3 i p_4, els temps d'arribada des de p_0 serien t=22 i t=23, mentre que des de p_1 aquests temps d'arribada seran t=20 i t=21. D'aquesta forma, el missatge arribarà als peers p_3 i p_4 a través del peer p_1.

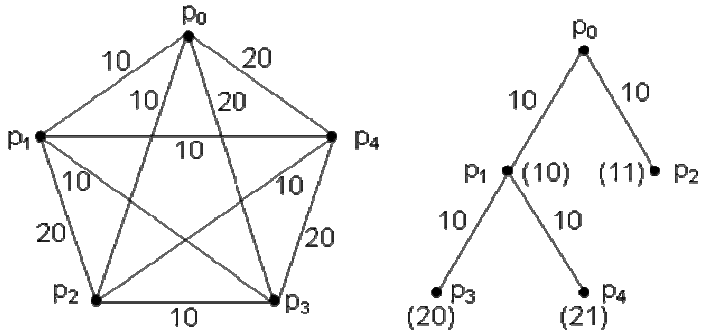


Fig. 3.5 Exemple de l'algorisme SMM en un graf complet

Es pot observar que si p_2 reenvia el missatge al peer p_4 el missatge també arribarà en t=21. La selecció d'un dels dos peers p_1 o p_2 per l'enviament del missatge a p_4 depèn de la utilització d'una comparació estricta o no en la línia on l'algorisme comprova si el node corresponent ha d'ésser escollit (ens referim a l'ús de "<" o "≤"). Aquesta consideració presenta un efecte en el grau dels peers dins l'arbre multicast, i per tant en la càrrega dels peers en termes de computació de xarxes. Encara que els efectes de càrrega de computació no són l'objectiu principal d'aquest estudi, sembla útil preservar el grau dels peers al mínim, sobre tot tenint en compte que a les xarxes *overlay* els peers corresponen als dispositius finals d'usuari.

Es pot provar que el temps multicast aconseguit per l'algorisme SMM és mínim quan μ_p=0 per a tots els peers. La prova és simple: en aquest cas, atès que μ_p=0 per a tots els peers, el retard entre dos nodes p i q és sempre el pes λ_{pq} de l'aresta que els uneix, i d'aquesta manera l'algorisme correspon a l'algorisme òptim de Dijkstra de complexitat $O(n^2)$.

Per un cas general, tanmateix, l'algorisme SMM no és sempre òptim. A la figura 3.6 es mostra una xarxa on l'algorisme SMM no és òptim. A l'esquerra s'ha aplicat el SMM amb el resultat d'un temps multicast de 7. A la dreta es mostra com el temps pot ser reduït a 5 redefinint els ordres de transmissió. En conseqüència, si la font comença amb el peer p_3, i segueix amb p_1 i finalitza amb p_2 el retard multicast serà de 5. En aquest dibuix es mostra entre parèntesi

el temps en què el paquet arriba a cada peer. El temps de transmissió és igual a la unitat per a tots els nodes.

En qualsevol cas, per a una xarxa *overlay* es pot assumir que $\mu_p << \lambda_{pq} \ \forall \ p,q$ i d'aquesta manera $\mu_p \approx 0$. Això significa que en una xarxa *overlay* l'algorisme SMM es quasi òptim (atès que hem vist que quan $\mu_p = 0$, l'algorisme SMM és òptim). A més, en el cas general en què $\mu_p \neq 0$, es pot trobar una solució òptima redefinint l'ordre de transmissió de tots els peers. De fet, és possible definir per cada peer tots els ordres de transmissió possibles del missatge i a continuació redefinir el pes de cada aresta, afegint a λ_{pq} el valor de $(i\text{-}1)\cdot \mu_p$, on *i* es l'ordre de transmissió des d'un peer *p* a un peer *q* en cada cas. Aleshores, podem aplicar l'algorisme de Dijkstra sobre tots els grafs resultants i escollir el millor retard de tots. Tanmateix, si es tracta d'un graf complet amb *n* peers, llavors, tindrem *(n-1)!* ordres de transmissió diferents per a qualsevol peer, que donarà, en combinació amb tost els peers, un nombre total de grafs de $((n\text{-}1)!)^n$, que en la pràctica no és computable per a valors alts de *n*.

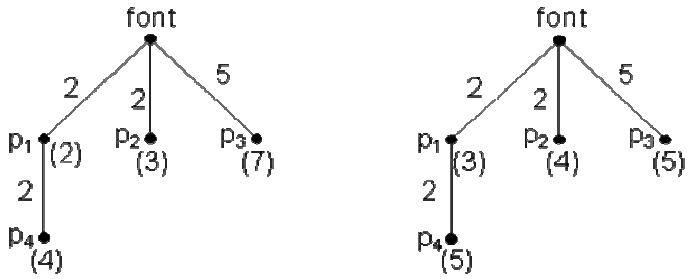


Fig. 3.6 Exemple d'una xarxa on l'SMM no és òptim

En aquest punt, es pot provar que la complexitat de l'algorisme SMM per *EMPS(n,λ,μ)* és de $O(n^2)$. Per cada iteració, l'SMM busca el peer que encara no ha rebut el missatge i que presenta el retard més baix. Com el nombre màxim de peers sense visitar és *n* aquesta operació requereix com a molt *n* comparacions. A més a més, com l'algorisme executa cada pas per a cada peer que ha rebut el missatge, el nombre total d'iteracions és igual al nombre de nodes. En conseqüència, es tenen *n* iteracions i a cada iteració fem un nombre màxim de *n* comparacions més algunes operacions bàsiques i acotades que presenten com a resultat una complexitat de $O(n^2)$.

3.4 *Message Stream Multicast Algorithm*

L'algorisme SMM ha estat definit per la transmissió multicast d'un únic missatge. Aquesta limitació suposa un inconvenient, ja que normalment es vol enviar més d'un missatge o paquet. D'aquest forma, l'algorisme SMM només té sentit en el cas que la diferència entre la producció de dos missatges consecutius sigui més gran que el temps multicast de transmissió d'un missatge. Si es considera, per exemple, una aplicació real de *streaming* de

vídeo, el que s'ha considerat com a missatge en *EMPS(n,λ,μ)* pot ésser el quadre (*frame* en anglès) d'un vídeo. En aquest cas, dos quadres de vídeo consecutius es generen amb una diferència de temps igual a l'inversa de la ràtio de quadres per segon (fps), que acostuma a prendre un valor entre 16 i 32 fps, depenent de la qualitat de vídeo. D'aquesta manera, la font ha d'enviar dos missatges consecutius en un interval de temps comprés entre 33 i 62,5 mil·lisegons.

La primera aproximació al *streaming* de missatges és la de repetir indefinidament la taula d'encaminament obtinguda amb l'algorisme SMM, fent multicast de cada missatge com si fos independent de la resta. Això significa que quan un missatge arriba finalment a tots els membres del grup multicast, la font començarà la transmissió multicast del següent missatge i així successivament. El retard total de multicast de l'*stream* T_M serà en aquest cas el nombre de missatges M multiplicat pel retard multicast SMM d'un únic missatge, que a partir d'ara anomenarem t_0. L'inconvenient principal d'aquesta solució és que la font no pot començar a enviar el següent missatge fins que tots els membres del grup han rebut l'anterior i això pot augmentar la cadència del missatge (tal com s'ha definit SMM, la font normalment retransmetrà el missatge fins al final de la comunicació, per tal d'inundar abans tot el grup multicast i minimitzar el retard final d'un únic missatge). Pel cas de la transmissió de vídeo, la pèrdua de cadència pot ésser solucionada mitjançant tècniques de *buffering*, però en tot cas afectarà el retard total de la comunicació.

A continuació, considerem una nova possibilitat per tal de fer multicast quan es vol transmetre més d'un missatge. Abans que el primer missatge arribi a tots els nodes, la font pot deixar d'enviar-lo i començar a transmetre el segon missatge. Amb aquesta condició, augmenta el retard amb què el primer missatge arriba a tot el grup. Però, d'altra banda, es comença abans la transmissió del segon missatge, i del tercer, i del quart i així successivament, de forma que el temps estalviat entre l'enviament de dos missatges consecutius serà progressivament acumulat, i si el nombre de missatges és prou gran, compensarà l'augment del retard t_0 del primer missatge.

L'algorisme modificat, al qual s'ha donat el nom de MSM-s *Message Stream Multicast Algorithm*, aturarà la transmissió del missatge un cop un peer hagi enviat el missatge s vegades. Aleshores començarà a enviar el següent i així successivament. L'algorisme aplica el mateix esquema per l'enviament de tots els missatges. Primer, enviarà el primer missatge, després el segon, i així successivament, amb la particularitat, que podrà començar a enviar el segon missatge abans que el primer hagi arribat a tots els nodes destí. La definició de MSM-s és la mateixa que la del SMM amb la restricció que els nodes només poden enviar el missatge com a molt s vegades i la particularitat que pot ser aplicat a missatges successius. D'aquí, l'esquema de l'algorisme de la figura 3.4 pot estendre's per l'algorisme MSM-s amb la diferència que ara la funció *imin()* escollirà el següent peer per transmetre dins del conjunt de peers que encara no han reenviat el missatge s vegades o, com es mostra a la secció 3.5, del conjunt de peers que han reenviat el missatge durant un temps inferior a un valor determinat.

La restricció del nombre de retransmissions pot aïllar alguns nodes si no disposem d'una connexió total, com es mostra a la figura 3.7 quan $s \leq 3$. En aquest cas, l'algorisme MSM-s hauria d'escollir un restricció mínima del nombre *s* que garanteixi que tots els peers rebin tots els missatges. A banda d'això, quan es restringeix el nombre de retransmissions de cada peer, MSM-s ha de tenir en compte la cadència del node font de generació de paquets (o misatges). Això és, si el peer font envia el primer missatge com a molt *s* vegades i després de $s \cdot \mu_r$ unitats de temps atura la transmissió i comença a transmetre el segon missatge, s'ha d'assumir que el node font té el segon missatge llest per a la transmissió en aquest instant (on μ_r és el seu temps de transmissió). Això significa que la cadència de la font ha de ser suficientment gran per proporcionar un nou missatge cada $s \cdot \mu_r$ unitats de temps. D'altra manera, la font aturarà l'enviament del primer missatge abans de tenir el segon llest amb un interval innecessari d'inactivitat, amb la conseqüent pèrdua d'eficiència. Per tant, l'MSM-s ha d'escollir una restricció mínima del nombre *s* no només per evitar l'aïllament de peers (com ja s'ha vist) sinó també per tal d'evitar la inactivitat de la font, de forma que $s \cdot \mu_r$ no sigui molt més petit que el temps de cadència dels missatges.

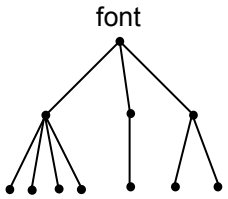


Fig. 3.7 Una xarxa on MSM-s aïlla un peer per a $s \leq 3$

Per una xarxa donada, si s'aplica MSM-s en comptes de MSM-(s+1) i si el nombre màxim de missatges enviats per qualsevol peer és *s* en comptes de *s*+1, aleshores el primer missatge arribarà més tard a tot el conjunt multicast de nodes destí, però es podrà començar a transmetre el segon amb anterioritat, i així el tercer i el quart, etc. En aquest cas, si el nombre de missatges és prou gran, l'increment de temps multicast per a un únic missatge t_0 serà compensat i l'MSM-s serà més ràpid que l'MSM-s+1. A [18] s'ha provat que sota certes condicions és possible calcular un nombre mínim de M_σ de tal manera que si el nombre de missatges és igual o més gran que M_σ aleshores MSM-σ és millor que MSM-σ+1.

Finalment, es pot provar que l'algorisme MSM-s per a *EMPS(n,λ,μ)* presenta una complexitat de $O(n^2)$. La prova és la mateixa que per l'SMM, atès que MSM-s realitza les mateixes operacions a cada iteració amb la diferencia que l'MSM-s ha de limitar a *s* el nombre de retransmissions per a cada peer. Aquesta petita restricció, tanmateix, no comporta cap efecte en la complexitat de MSM-s. L'algorisme executa *n* iteracions i a cada iteració realitza un màxim de *n* comparacions més algunes operacions acotades, incloent les limitacions en el nombre de retransmissions. Així, la complexitat de MSM-s és de $O(n^2)$.

3.5 L'algorisme MSM amb restricció de temps

A la figura 3.8 es mostra un peer *q* que té el mateix temps de transmissió μ_p que un peer *p* que reenvia el missatge a *q*. Suposem que $s_p(s) \leq s$ és el nombre de vegades que *p* reenvia el missatge en l'MSM-s. En aquest cas el segon missatge serà rebut per *q* amb un retard de $s_p(s)\cdot\mu_p$ unitats de temps respecte el primer missatge, ja que el segon missatge segueix el mateix camí amb un retard d'origen igual $s_p(s)\cdot\mu_p$. És a dir, *p* enviarà el primer missatge $s_p(s)$ vegades i llavors, $s_p(s)\cdot\mu_p$ unitats de temps més tard, començarà a enviar el segon i així successivament. Si en aquest cas també es limita a $s_q(s)\cdot\mu_q$ el nombre de retransmissions del node *q*, aleshores *q* rebrà el segon missatge al mateix temps que acaba d'enviar el primer missatge per última vegada. Encara que això no és important ja que normalment es disposa de connexions *full-duplex*, es pot evitar restringint a $s_q(s)=s_p(s)-1$ el nombre de retransmissions del peer *q*.

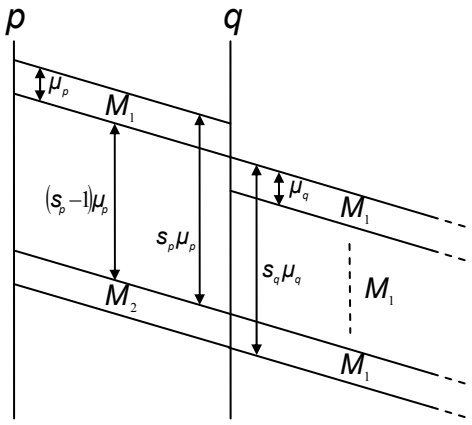


Fig. 3.8 Transmissió de dos peers diferents

Per un cas més general en què els peers poden tenir diferents temps de transmissió, podem veure què passa quan el temps $s_q(s)\cdot\mu_q$ d'enviament d'un missatge del peer *q* és més gran que el període de reenviament $s_p(s)\cdot\mu_p$ del peer *p* que es troba en un nivell superior de l'arbre multicast (per exemple, podem suposar que *p* reenvia el missatge a *q*). En aquest context, el segon missatge pot arribar al peer *q* abans que aquest hagi acabat de transmetre el primer missatge, i aleshores el segon missatge s'ha d'emmagatzemar en un *buffer*, amb el consegüent retard. Aquest retard s'acumularà amb el tercer missatge, i amb el quart i així successivament. Aquesta situació es pot evitar limitant el període de temps $s_q(s)\cdot\mu_q$ durant el qual cada peer reenvia el missatge. Per això, forçarem l'algorisme de tal forma que la cadència de reenviament $1/(s_q(s)\cdot\mu_q)$ de qualsevol peer *q* sigui més gran que la cadència $1/(s_p(s)\cdot\mu_p)$ de qualsevol peer *p* que estigui (dins de l'arbre multicast) en el camí entre la font i el peer *q*. Això equival a dir que $s_q(s)\cdot\mu_q < s_p(s)\cdot\mu_p$.

D'altra banda, no interessa que $s_p(s)\cdot\mu_p >> s_q(s)\cdot\mu_q$ ja que en aquest cas el node *q* aturaria la retransmissió del primer missatge molt abans de rebre el segon, cosa que comportaria una pèrdua d'eficiència de l'algorisme (o de desaprofitament d'ample de banda). D'aquesta manera, el valor de $s_q(s)$ ha de ser diferent per a cada node *q*, depenent del seu temps de transmissió μ_q i també dels paràmetres $s_p(s)$ i μ_p dels peers anteriors a ell en l'arbre multicast, de tal manera que $s_p(s)\cdot\mu_p > \approx s_q(s)\cdot\mu_q$. Com el node font o *root* ocupa la cúspide de l'arbre multicast, també es tindrà per a qualsevol peer *q* la desigualtat $s\cdot\mu_r \geq s_r(s)\cdot\mu_r > \approx s_q(s)\cdot\mu_q$.

3.6 L'algorisme MSM amb restricció de cadència

Com l'algorisme MSM-s depèn d'alguns paràmetres (fonamentalment: el nombre de retransmissions del node origen i el temps màxim de transmissió d'un missatge per a qualsevol dels altres peers), es poden aplicar certes modificacions sobre l'algorisme original MSM-s, per tal de millorar el seu comportament en termes de congestió. Així, tots els algorismes definits en el present treball estan basats en la definició de la figura 3.4 amb alguns canvis en la funció *imin()* que escull el següent peer que ha de reenviar el missatge. De fet, aquests canvis afecten principalment dos aspectes: la condició per considerar un node millor que un altre i l'estratègia que seguim quan, sota certes condicions, l'algorisme no pot trobar cap node per enviar el missatge (i, per tant, l'arbre multicast es trunca abans d'arribar a tot el grup i no s'assoleix connectivitat). A continuació, s'exposen els algorismes plantejats en treballs anteriors i, a l'últim punt, l'algorisme presentat en aquest treball.

Cadence: amb aquest nom es defineix l'algorisme MSM amb restricció de temps, com el que s'ha definit a la secció anterior. En aquest cas, el temps límit *b0* és el valor de *s* multiplicat per la mitjana de tots els temps de transmissió del node font, també anomenat arrel (s'ha de notar que els temps de transmissió entre l'arrel i els altre peers no és sempre el mateix, sinó que depèn de l'ample de banda dels diferents enllaços punt a punt que uneixen l'arrel amb els altres nodes). El node font pot enviar cada paquet com a molt *s* vegades, mentre que la resta de peers podran reenviar el paquet sempre i quan el seu temps de cadència no superi el límit de temps *b0*, això es, sempre i quan el seu temps total de transmissió d'un missatge sigui inferior a *b0*. S'ha de notar que amb aquest algorisme, depenent del valor de *s* i l'ample de banda dels enllaços, alguns peers poden romandre aïllats (per exemple si un peer té un accés molt lent i el temps per transmetre el missatge cap a ell es superior a *b0* des de qualsevol altre peer), i podem obtenir finalment un arbre incomplet. Per resoldre això, augmentarem el valor de *s*. També es poden trobar casos on el node acompleixi la seva condició de cadència, però aquest temps sigui superior al temps de cadència de l'arrel. Per exemple, sigui *s*=5 i el valor de *b0* de 50 ms (ja que el promig de μ per l'arrel és de 10). Llavors, si el node arrel envia un missatge als peers més propers, on μ és 9 ms per a tots ells, es tindrà un temps de cadència per l'arrel de 45 ms. Com el límit de temps és *b0*, algun altre node pot tenir un temps de cadència de 48 ms, que és més petit que *b0* però més gran que el període de transmissió de l'arrel, extrem que pot comportar problemes de congestió.

En aquesta variació de l'algorisme, el node arrel pot enviar el paquet com a molt *s* vegades, en comptes d'utilitzar el límit de temps *b0*, ja que aquesta cota podria comportar un arbre més restrictiu. Per explicar això, plantegem el següent exemple, amb valors de *s*=5 i *n*=10. Suposem que el node arrel necessita un temps de transmissió de 1, 2, 2, 4, 10, 2, 2, 2, 2, i 2 ms per enviar el paquet als altres peers, ordenats per proximitat (en temps de transmissió més temps de propagació). En aquest cas, el límit *b0* serà de 14,5 ms. Llavors, si com s'ha dit, l'arrel pot enviar el paquet almenys *s* vegades, s'obté que el seu temps de cadència pot ser de 1+2+2+4+10=19 ms (considerant que per a qualsevol d'aquets peers, qualsevol camí entre un node visitat i un altre té un cost superior a l'enllaç de l'arrel). Si, d'altra banda, es limita la cadència del node arrel al temps *b0*, es pot observar que quan l'arrel hagi enviat el paquet 4 vegades, el seu temps de cadència serà de 1+2+2+4=9 < *b0*, però si envia el paquet una vegada més, la seva cadència serà més gran que *b0* (19 > 14,5). Aleshores, el node arrel no podrà enviar el cinquè paquet. Per això, finalment *b0* limita tots els peers fora de l'arrel, que podrà retransmetre un missatge fins a *s* vegades.

Forced leaves: en aquest cas, l'arrel pot enviar el paquet com a molt *s* vegades, i també es determina el valor de *b0* com en el cas anterior. Un altre cop, els nodes poden enviar el paquet sempre i quan el seu temps de cadència (és a dir, el temps durant el qual retransmeten un mateix missatge a diferents peers) sigui igual o inferior al límit *b0*, però aquesta vegada, si algun node no acompleix la condició de retransmissió, i encara queden nodes aïllats, l'algorisme força a un o més nodes (que seran escollits entre els que encara no hagin enviat el missatge) a enviar el missatge una vegada. Aquests nodes són escollits, com sempre, amb l'objectiu de minimitzar el temps d'arribada del missatge al següent node aïllat. Aquest algorisme sempre genera arbres complets, però ara alguns nodes poden superar el límit *b0* i igualment el temps de cadència del node arrel, amb els consegüents problemes de congestió.

Forced all peers: aquest algorisme és molt similar a l'anterior, però en aquest cas, si encara queden peers per rebre el missatge, es pot forçar a qualsevol node a reenviar la informació (en el cas anterior només es forçaven els nodes que encara no havien retransmès el paquet). Una altra vegada, el peer que reenvia el missatge es escollit en funció del temps d'arribada al següent peer aïllat. Els resultats són similars a l'anterior amb problemes de congestió quan certs nodes superen el límit *b0* i el temps de cadència de l'arrel.

Optimum t0 (optt0): aquest algorisme no utilitza el límit *b0*. El node arrel pot enviar el paquet com a molt *s* vegades, mentre que la resta de nodes poden reenviar el paquet sempre i quan el seu temps de cadència no superi el temps de cadència de l'arrel en aquell moment. En aquest cas, l'arbre resultant pot ésser inconnex (per evitar-ho anirem augmentant *s* fins a obtenir connectivitat total) i el temps total de transmissió d'un missatge t_0 és més elevat que en els casos anteriors. Tanmateix, no es presenten problemes de congestió, ja que el node font tindrà el temps de cadència més elevat de tota la xarxa i qualsevol altre node haurà acabat d'enviar un missatge abans de rebre el següent.

Fins aquí hem exposat els algorismes desenvolupats i estudiats a [18]. A partir dels resultats obtinguts, proposem a continuació un nou algorisme amb l'objecte de disminuir el màxim temps de cadència de tots els peers de la xarxa i, per tant, d'augmentar la cadència amb què enviem els successius missatges. Com s'ha vist en apartats anteriors, aquest paràmetre determina a mig i llarg termini el retard principal de la transmissió.

Optimum cadence (optcad): aquest algorisme és molt similar a l'anterior, però aquesta vegada, i a diferència de la resta d'algorismes, canvia el criteri de selecció del següent node que envia el missatge. En aquest cas, els peers s'escullen en funció del seu temps total de transmissió (d'un missatge), sense tenir en compte el temps de propagació. És en aquest punt on es canvia el criteri d'optimització: ara es busca minimitzar el temps durant el qual un peer transmet un mateix missatge a d'altres peers, en comptes de minimitzar el temps d'arribada al peer següent. Aleshores, el node arrel podrà enviar com a molt *s* vegades, no existeix el límit *b0*, i la resta de peers podran retransmetre el paquet sempre i quan el temps total de transmissió sigui inferior al temps de cadència de l'arrel. Com en el cas anterior, l'arbre resultant pot ser incomplet i no es presentaran problemes de congestió, ja que qualsevol node haurà acabat de transmetre el missatge abans de rebre el següent. En el capítol de resultats veurem, en tot cas, com aquest algorisme es pot aplicar sense restringir el temps de cadència de cap node, per tal de disminuir el retard total de la transmissió.

4. IMPLEMENTACIÓ DELS ALGORISMES

Als capítols anteriors s'han descrit els aspectes fonamentals dels algorismes d'encaminament que són l'objecte d'aquest treball. En tot cas, és necessari mostrar l'execució dels algorismes, és a dir, comprovar el seu comportament sobre una xarxa real. Per realitzar aquesta tasca, s'han executat els algorismes sobre el model de xarxa d'Internet presentat a la secció 1.3, a la qual hem afegit un conjunt de nodes d'usuari, anomenats peers. A continuació s'explica el procediment que hem seguit per afegir aquest conjunt d'usuaris, obtenir la xarxa *overlay* i aplicar els algorismes sobre la mateixa.

4.1 Generador del graf de xarxes

Com ja s'ha comentat a les seccions anteriors, el primer que hem de fer per simular una xarxa *overlay* és generar la xarxa troncal. Per això hem considerat prou bo el model de *Transit-Stub* descrit a la secció 1.3, que depèn d'un conjunt de variables. Aquest model pot ésser implementat fàcilment amb el paquet de la *Georgia Tech Internet Topology Models GT-ITM*, un conjunt d'aplicacions que permeten dibuixar grafs amb diferents topologies segons una varietat de lleis de probabilitat. També permeten dibuixar grafs seguint els models presentats a la secció 1.2. Aquest paquet treballa amb un arxiu generador de dades, que conté tots els paràmetres de la xarxa troncal. La sortida es un arxiu ".gb" amb la informació del graf (nodes, arestes i pesos). Encara que es tracta d'un arxiu de text, no és fàcil de llegir a causa de la seva descripció dels nodes i les arestes. Per tal de resoldre aquesta dificultat, s'ha utilitzat una altra eina del paquet, que transforma els arxius ".gb" en arxius ".alt" amb informació molt més intel·ligible. Aquest arxiu s'ha fet servir amb una de les aplicacions per afegir els nodes d'usuari o peers, generant el graf final d'usuaris. El paquet de la *GT-ITM* posseeix més eines. Per exemple, l'arxiu ".gb" pot ser utilitzat amb el simulador NS-2 *Network Simulator*; es poden igualment calcular determinats paràmetres de la xarxa resultant; i fins i tot es pot fer servir una altra aplicació, el *NED Network Editor*, per representar gràficament la xarxa troncal.

La topologia del graf resultant és similar a la topologia real d'Internet, amb un conjunt de nodes actuant com a routers de la xarxa troncal, i amb enllaços d'alta velocitat que interconnecten les subxarxes (o *stubs)*. Aquestes subxarxes representen les xarxes ISP, o LANs corporatives, que disposen de connexions amb la xarxa troncal. Finalment, s'han d'afegir a les xarxes ISP els peers d'usuari, és a dir, els nodes que formaran el grup multicast.

4.2 Aplicacions de simulació dels algorismes MSM

4.2.1 Descripció de les aplicacions

Per tal de dissenyar la xarxa *overlay*, aplicar els algorismes sobre ella i obtenir els resultats finals, s'ha treballat amb una aplicació creada en projectes

anteriors [7,17,18] i, a més a més, s'han creat nous codis de programa per treballar amb més comoditat i fluïdesa. L'aplicació, anomenada "MSM Algorithms Simulator" i representada a la figura 4.1, s'ha utilitzat en el present estudi per a la generació tant dels grafs troncals com dels nodes d'usuari.

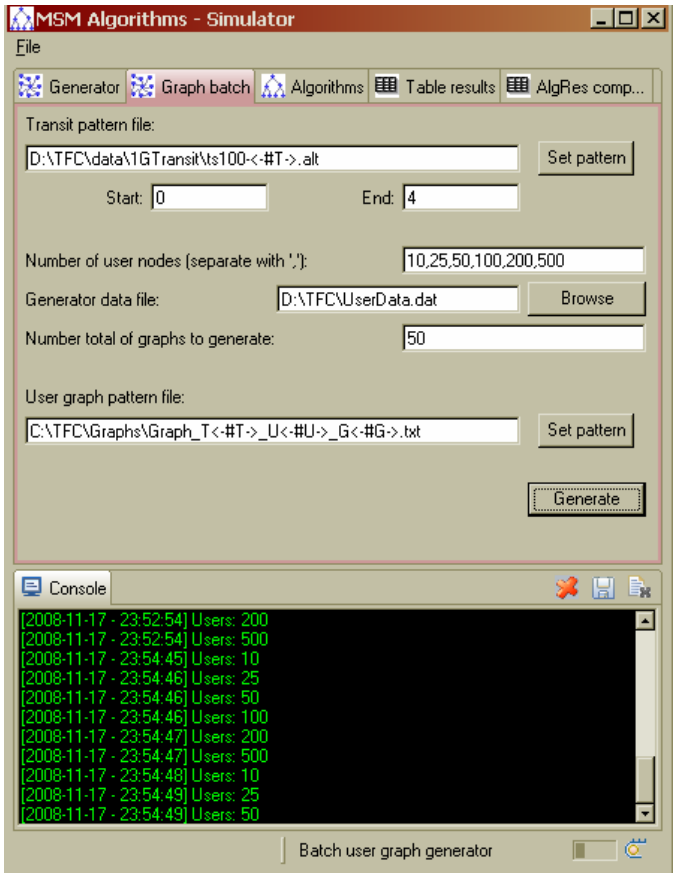


Fig. 4.1 Aplicació MSM Algorithms Simulator

Després, s'han desenvolupat dues aplicacions més. La primera és una programació de l'algorisme de Dijkstra en llenguatge Java, per tal d'obtenir els grafs *overlay* i poder aplicar-hi els algorismes. Aquesta aplicació dóna com a resultat un graf *overlay* que només considera els nodes d'usuari amb les corresponents variables de temps necessàries: temps de transmissió i temps de propagació, per a tots els parells de peers.

La segona aplicació creada permet llegir els grafs *overlay* creats amb les aplicacions anteriors i fer-hi córrer els algorismes sobre ells. Aquesta aplicació ha estat programada en C seguint uns criteris que simplifiquen la modificació de variables, funcions i la manipulació d'arxius, entre d'altres. D'aquesta manera es poden alterar fàcilment els algorismes per tal de millorar els retards finals de transmissió.

Els motius pels quals s'han creat aquests nous programes i no s'ha continuat amb l'aplicació creada a [18] són bàsicament els següents. Primer, per

l'excessiva modularitat que presentava l'aplicació MSM Algorithms Simulator, la qual cosa complicava de manera excessiva la implementació de petites modificacions. La segona és el fet que amb aquella aplicació mai es podien obtenir els grafs *overlay* pròpiament dits, és a dir, la xarxa *overlay* s'anava creant a mida que s'executaven els algorismes i no es desava enlloc. Això comporta un desavantatge quan treballem amb nous algorismes, ja que no només s'ha de dur a terme la modificació de l'algorisme, sinó que també s'ha de crear novament el graf *overlay* a mesura que avança la seva execució. Finalment, la tercera raó ha estat la despesa de temps que comportava l'execució dels algorismes, atès que estaven programats en Java, llenguatge molt més lent que el C amb què hem programat la nova aplicació.

4.2.2 Generació dels grafs d'usuari

Com s'ha dit, per crear la xarxa *overlay* primer de tot hem de generar el graf troncal de la xarxa i a continuació afegir els usuaris del grup multicast. Aquesta segona tasca es duu a terme amb l'aplicació desenvolupada a [18], que treballa amb un arxiu ".alt" que conté la xarxa troncal (definida ja en projectes anteriors [7,17,18]). El mètode afegeix *n* nodes (que actuaran com a membres del grup multicast) connectant cadascun d'ells a un, i només un, node *stub*.

Aquesta funció necessita com a dades: un graf de trànsit (arxiu ".alt"), el nombre total de nodes d'usuari i un arxiu generador de dades (arxiu ".dat"). Aquest arxiu ".dat" conté les característiques dels nodes d'usuari que conformaran el grup multicast, com l'ample de banda dels enllaços (també si son simètrics o asimètrics), i el percentatge de nodes de cada tipus. Aquesta informació estableix els paràmetres de l'aplicació, la qual finalment afegeix cada peer: primer escull de manera aleatòria el node de la xarxa troncal entre els nodes *stub* (és a dir, prescindint dels nodes de trànsit), amb una distribució uniforme. D'aquesta manera ens assegurem que tots els nodes d'usuari estiguin connectats a una xarxa d'accés (tal i com es va dir al primer capítol, la xarxa troncal es divideix en nodes de trànsit i nodes *stub*, i aquests segons són els que representen les xarxes d'accés). Un cop tenim el node *stub*, s'afegeix l'enllaç del nou usuari, amb un ample de banda aleatori seleccionat uniformement de l'arxiu d'informació (".dat"). En aquest arxiu es troba la informació sobre les probabilitats de l'ample de banda dels usuaris. El tipus d'informació (velocitat en kbps o Mbps, enllaç simètric o asimètric, nombres absoluts o percentatge) es fixat a través d'altres paràmetres del generador. Aquesta aplicació ens dóna com a resultat uns arxius ".grf" que inclouen, afegits als nodes inicials de la xarxa troncal (formada per 100 nodes en total), el nombre de peers indicat per l'usuari. A partir d'aquí ja podem utilitzar les següents aplicacions per tal d'obtenir el graf *overlay* dels usuaris.

4.2.3 Obtenció dels grafs *overlay*

El següent pas, com s'ha dit, és l'obtenció dels grafs *overlay*. Per dur a terme aquesta tasca s'ha realitzat una aplicació que a partir de la xarxa completa (un arxiu ".grf") obté la xarxa *overlay*, és a dir, un graf format només pels nodes

d'usuari i les "distàncies" que els separen (temps de transmissió i de propagació). Els arxius resultants, arxius ".ovr", permetran executar els algorismes d'encaminament i comparar i analitzar els seus retards.

Aquesta aplicació, llegeix l'arxiu generat amb anterioritat (el ".grf") i aplica l'algorisme de Dijsktra sobre els nodes d'usuari (això per calcular el temps de propagació entre dos usuaris, que és la suma del temps de propagació de tots els enllaços intermedis; d'altra banda, el temps de transmissió entre dos usuaris el dóna l'enllaç més lent). El resultat és una matriu ordenada de nodes: per a cada node tenim els veïns ordenats (segons el temps de transmissió més la propagació; o només segons el temps de transmissió, depenent de l'algorisme aplicat), amb els seus temps de transmissió, propagació i retard total.

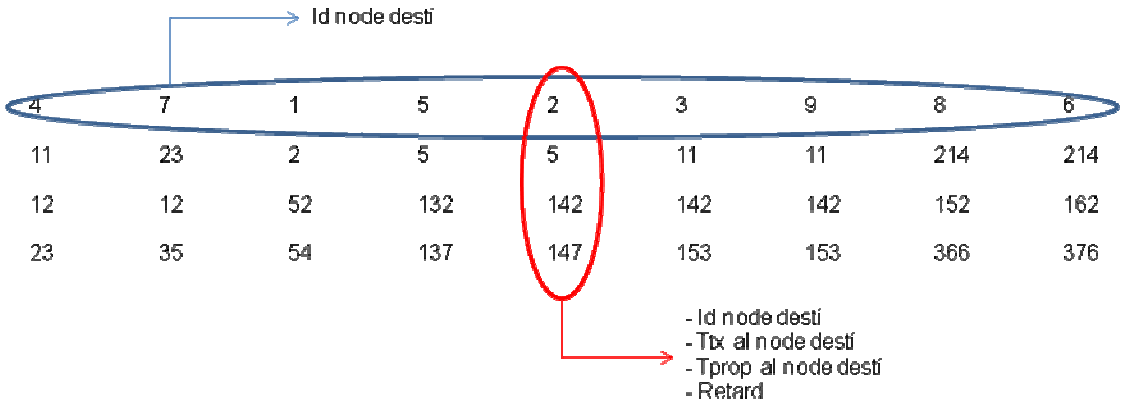


Fig. 4.2 Exemple d'un node de la xarxa *overlay*

4.2.4 Simulació dels Algorismes

Un cop definida la xarxa *overlay*, s'han d'aplicar els algorismes sobre ella. Per realitzar aquesta tasca, s'ha desenvolupat una aplicació en C, que executa els algorismes sobre els grafs i torna com a resultat un arxiu ".res" amb diferents dades, com els arbres multicast resultants, els retards obtinguts o els nodes amb risc de congestió.

Aquesta aplicació fixa primer de tot el node arrel (que sempre és el 100, és a dir, el primer dels nodes d'usuari generats, ja que els nodes que van del 0 al 99 pertanyen a la xarxa troncal). L'aplicació necessita conèixer els diferents valors de *s* que ha d'aplicar (que variaran segons el nombre d'usuaris), a més de, com és obvi, l'especificació de l'algorisme que s'executa en cada cas (escollit entre la llista presentada al final de l'anterior capítol). En el nostre cas, hem programat els algorismes en un arxiu de text amb extensió ".c" que hem compilat i executat sobre el sistema operatiu Linux.

5. AVALUACIÓ DELS ALGORISMES

En aquest capítol es descriuen els paràmetres amb què hem definit les xarxes *overlay*. A continuació apliquem sobre aquestes xarxes els dos darrers algorismes presentats a la secció 3.6 (optimum t0 i optimum cadence, que abreujarem com a optt0 i optcad) i presentem els resultats en termes de retard i nombre total de nodes amb risc de congestió. El motiu pel qual s'apliquen els algorismes optt0 i optcad, és que a [18] optt0 és el millor dels algorismes proposats i, d'altra banda, optcad pretén ser-ne una millora.

5.1 Parametrització de la xarxa de trànsit

Tal i com s'ha especificat a la secció 4.1, el primer pas per simular una xarxa *overlay* és definir el seu nucli, o el que hem anomenat també xarxa troncal. Amb aquest propòsit, s'han utilitzat les eines *GT-ITM* per construir cinc nuclis de xarxa d'acord amb els paràmetres de la taula 5.1. Això representa 5 nuclis de xarxa diferents basats en el model *Transit-Stub*.

Taula 5.1. Parametrització de la *GT-ITM*

Transit domains	1
Av. Nodes (T) / Transit domain	4
Stub domains / Transit node	3
Av. Nodes (S) / Stub domain	8
Edge Prob (between nodes in the same Transit domain)	0.6
Edge Prob (between nodes in the same Stub domain)	0.42
Transit-Stub extra edges	0
Stub-Stub extra edges	0
Transit-Transit bandwidth	1 Gbit/s
Transit-Transit propagation delay	100 ms
Transit-Stub bandwidth	100 Mbit/s
Transit-Stub propagation delay	10 ms
Stub-Stub bandwidth	100 Mbit/s
Stub-Stub propagation delay	10 ms

5.2 Generació dels grafs

Per a l'obtenció d'una xarxa *overlay* per provar els algorismes, com a complement del model *Transit-Stub*, s'ha aplicat el generador descrit a la secció 4.2. En aquest punt s'han considerat els paràmetres d'usuari apareguts en un informe de febrer del 2006 publicat per l'AIMC (Asociacion para la investigación de Medios de Comunicación) [1]. Encara que podríem haver pres una taula més actualitzada, hem treballat amb aquesta per tal de poder comparar els nostres algorismes amb els resultats dels treballs anteriors. Els

peers representen usuaris ADSL connectats a una xarxa ISP. Els valors utilitzats a l'arxiu generador de dades són els mostrats a la taula 5.2.

Hem considerat igualment que el retard de propagació dels enllaços que connecten els peers a un node *stub* és de 1 ms, ja que normalment els nodes d'usuari i els nodes d'accés es troben molt propers els uns dels altres. A la taula, les velocitats de l'ample de banda són simètriques, a diferència del que passa amb ADSL. En tot cas, l'increment progressiu de les velocitats de pujada juntament amb la utilització dels enllaços d'alta velocitat, com l'ús futur de FTTH (de l'anglès *Fiber To The Home*), fan aquesta restricció vàlida per a la prova dels algorismes. Per calcular els temps de transmissió hem suposat que enviem paquets o missatges de 1500 bytes de longitud. Amb aquests valors, s'han generat 10 grafs *overlay* per a cadascuna de les cinc xarxes troncals amb 10, 25, 50, 100, 200 i 500 usuaris respectivament (un total de 50 xarxes per a cada nombre de nodes d'usuari).

Taula 5.2. Informe AIMC de febrer de 2006

Enllaç base	Usuaris absoluts	%
No se sap	3.725	7,1
56 kbps	4.951	9,4
64 kbps	445	0,8
128 kbps	914	1,7
256 kbps	2.235	4,3
512 kbps	5.278	10,1
1 Mbps	21.811	41,6
2 Mbps	6.342	12,1
4 Mbps	5.767	11,0
8 Mbps	765	1,5
No saben/No contesten	165	0,3

5.3 Resultats i comparacions

Per a cada graf *overlay*, s'han aplicat els dos algorismes optt0 i optcad descrits a la secció 3.6, amb *s* des de 1 fins a *n* si el nombre d'usuaris *n* és menor o igual a 25, i fins a 30 si el nombre d'usuaris es més gran. Després, per a cada algorisme i cada valor de *s*, s'obtenen els resultats de cada execució: connectivitat total o no, valor mínim de *s* per aconseguir connectivitat total (fila s_{min}), màxim temps de cadència t_{cad} en tot l'arbre, temps de cadència del node arrel (que no hem apuntat perquè al final ens interessa t_{cad}) i el retard t_0 de transmissió (incloent-hi la propagació) d'un únic missatge. També s'ha comptat el nombre de nodes que poden tenir problemes de congestió (fila "afectats") en el cas que algun node superi el temps de cadència de la font (els nodes afectats són els que superen el temps de cadència de la font, més els nodes que pengen dels anteriors a l'arbre multicast, ja que si un node pateix congestió o pèrdua de missatges, aquesta pèrdua afectarà igualment als nodes que han de rebre el missatge a través seu). Igualment, s'ha apuntat el mínim valor de *s*

que dóna el retard mínim (fila s_{best}). Com l'algorisme ha d'enviar els missatges a una cadència marcada pel temps de cadència més gran de tots els peers (per tal d'evitar congestió), considerem el retard que hem de minimitzar com aquest temps de cadència t_{cad} més gran. En el cas que dos o més resultats tinguin la mateixa cadència, aleshores considerem el mínim retard multicast t_0. Finalment, si encara tenim dos o més execucions amb la mateixa cadència i amb el mateix temps t_0, aleshores es consideren el nombre de nodes afectats.

Taula 5.3. Resultats pels algorismes optcad i optt0

		usuaris	10	25	50	100	200	500
Amb restriccions	opt cad	t_{cad}	212,44	295,04	306,86	341,8	343,8	335,82
		t_0	901,94	1531,64	2489,16	3856,72	5298,98	10830,3
		best	23	20	28	17	22	19
		S_{min}	2,6	3,86	3,9	4,5	4,9	4,64
		S_{best}	2,6	3,86	3,9	4,5	4,9	4,64
	opt t0	t_{cad}	226,7	287,94	328,52	326,54	332,72	336,12
		t_0	481,14	536,98	562,4	522,14	582,06	605,76
		best	12	21	16	26	23	24
		S_{min}	4,08	6,3	6,22	7,48	8,34	7,62
		S_{best}	4,08	6,3	6,22	7,48	8,34	7,62
	opt cad vs. opt t0	M_{max}	34,09	119,667	284,25	336,6	2131,67	967
		M_{min}	0,524	8,472	5,767	56,304	40,217	124,322
		L_{max}	0,05	0,171	0,407	0,48	3,049	1,383
		L_{min}	0,001	0,012	0,008	0,08	0,058	0,178
		mitjana M	11,309	27,67	84,88	142,21	421,16	430,74
		mitjana L	0,016	0,03	0,12	0,2	0,6	0,62
Sense restriccions	opt cad	t_{cad}	176,94	208,48	214	214	214	214
		t_0	1311,2	2843,32	4951,1	8246,6	13565,5	27893,8
		best	32	27	47	48	50	50
		S_{min}	1	1	1	1	1	1
		S_{best}	1	1	1	1	1	1
		afectats	3,28	9,1	16,26	35,02	85,3	190,36
	opt t0	t_{cad}	194,04	220,8	262,06	262,26	270,78	283,86
		t_0	471,56	539,92	558,92	544,12	602,2	593,42
		best	0	0	0	0	0	0
		S_{min}	1	1	1	1	1	1
		S_{best}	3,14	3,8	4,22	4,44	6,58	4,08
		afectats	5,24	14,62	39,28	86,44	170,58	453,24
	opt cad vs. opt t0	M_{max}	184,4	1273,5	490,55	1507,25	736,25	829,588
		M_{min}	1,473	43,767	15,696	73,41	85,667	120,883
		L_{max}	0,264	1,822	0,7	2,156	1,053	1,187
		L_{min}	0,002	0,063	0,022	0,105	0,121	0,173
		mitjana M	58,066	243,12	154,617	255,105	263,72	436,76
		mitjana L	0,083	0,347	0,221	0,364	0,377	0,624

La taula 5.3 recull els resultats en aplicar els dos algorismes optt0 i optcad. Per cada possible nombre *n* d'usuaris (10, 25, 50, 100, 200 i 500) s'han executat els dos algorismes sobre cinquanta grafs (cinc xarxes troncals sobre les que hem definit cada vegada deu xarxes *overlay*, cinquanta en total per a cada nombre *n* d'usuaris). Els valors presentats són, per a cada valor de *n*, el promig dels resultats sobre els cinquanta grafs *overlay* (excepte pels valors de *M* i *L*, que s'expliquen més endavant). Els temps s'expressen en mil·lisegons.

La taula té dues parts. En la de dalt (assenyalada "amb restriccions") hem imposat sobre els dos algorismes optt0 i optcad les restriccions descrites a la secció 3.6 per tal d'evitar congestió, és a dir, hem escollit només els peers que tornaven un temps de cadència més petit que el temps de cadència de la font. D'aquesta forma no hi ha cap peer "afectat", és a dir, en perill de congestió. A la segona part de la taula (assenyalada "sense restriccions") hem repetit els resultats obligant en cada cas al peer font a enviar com a molt *s* vegades un missatge, però sense imposar ara cap restricció a la cadència dels peers. D'aquesta forma, com els algorismes han tingut més llibertat, han trobat arbres amb retard menors, com es pot comprovar a la taula. El preu que paguem és l'aparició de nodes afectats per congestió. En tot cas, això és un desavantatge menor, ja que es pot resoldre si obliguem a la font a enviar els missatges amb una cadència igual a la del peer més lent de l'arbre (és a dir, si la separació entre dos missatges consecutius enviats per la font és igual al temps de cadència màxim de tota la xarxa; recordeu que el temps de cadència d'un peer l'hem definit com el temps durant el qual el peer retransmet cada missatge a d'altres peers). En aquest cas, els retards no variaran (ja que el retard màxim entre dos missatges consecutius el fixa el màxim temps de cadència de la xarxa) i a més cap peer rebrà un missatge abans d'haver retransmès l'anterior, com s'ha discutit en seccions anteriors. És a dir, estem evitant la possible congestió de la xarxa.

D'aquesta forma, treballarem sempre amb els algorismes sense restriccions (part de sota de la taula), tot imposant a posteriori, com hem dit, una cadència a la font igual a la cadència del node més lent. En aquest context, com es desprèn de la fila t_{cad} de la taula, sempre obtindrem millors resultats. Els retards, a més, no augmenten de forma considerable en funció del nombre de nodes de la xarxa. Això es deu a que si incrementem el nombre de peers augmenta el nombre de nodes als quals hem d'enviar la informació, però igualment augmenten els recursos de què disposem. En tot cas, el creixement del retard mai és proporcional al creixement de la xarxa. Això vol dir que els algorismes proposats poden aplicar-se sobre xarxes arbitràriament grans, ja que els seus retards no augmentaran de forma indefinida i, a més a més, la seva complexitat és petita $O(n^2)$ i per tant la seva execució no consumirà grans temps de computació. D'aquesta forma, en un context de xarxes amb connexions fortament variables, podríem executar periòdicament els algorismes per actualitzar les taules d'encaminament, en funció de l'estat de les connexions.

Com hem dit, la taula inclou els valors de S_{min} i S_{best} (valor de *s* mínim per aconseguir connectivitat, d'una banda, i valor mínim de *s* que torna el retard òptim, d'una altra). Per als algorismes sense restriccions (part de sota de la

taula) tenim que S_{min} és sempre igual a u, perquè com no limitem la cadència de cap node fora de la font, aleshores sempre podrem cobrir tota la xarxa. D'altra banda, per optcad el valor de S_{best} també sempre és igual a la unitat. Com l'algorisme tracta de minimitzar t_{cad}, aleshores tots els nodes enviaran un missatge el mínim nombre possible de vegades, normalment una, i això val també per la font. Aleshores els resultats no milloraran si deixem enviar a la font més d'un paquet, perquè normalment només voldrem que n'enviï un. D'altra banda, S_{best} és per optt0 més gran que la unitat. Això pot explicar-se perquè com optt0 tracta de minimitzar t_0 en comptes de t_{cad}, i els peers diferents de la font no tenen cap restricció, llavors algun d'aquets peers pot carregar-se amb un temps de cadència molt alt quan s=1, temps que pot disminuir quan deixem a la font enviar més d'un missatge.

Si passem a la part de dalt de la taula, els algorismes amb restriccions tenen un S_{min} més gran que la unitat. Això es deu a que si només deixem enviar un cop a la font i restringim a més el temps de cadència de tots els peers segons la cadència de la pròpia font, no tindrem prou temps de transmissió disponible per arribar a tota la xarxa. Per això, hem d'alleugerir les restriccions augmentant el nombre de vegades que la font i, consegüentment la resta de peers, poden enviar un mateix missatge, és a dir, incrementant el valor de s. D'altra banda, un cop arribem a tots els nodes, els algorismes ja no canvien d'estratègia i encara que deixem enviar més cops un missatge als diferents peers no ho faran, perquè d'aquesta manera augmentarien el seu temps de cadència. Per això en aquest cas S_{best} sempre pren el mateix valor que S_{min}.

D'altra banda, a la part de dalt de la taula dels algorismes amb restriccions, S_{min} i S_{best} són més petits per optcad que per optt0. Amb optcad els nodes normalment escolliran destinataris amb un temps de transmissió petit, de tal forma que amb un temps de cadència total limitat, podran enviar el missatge a més veïns. Aleshores, en comparació amb optt0, podrem obrir més fàcilment l'arbre multicast (és a dir, tindrem nodes amb graus superiors) i cobrir finalment tota la xarxa. D'aquesta forma, podrem arribar a tots els nodes amb una s més petita, tal i com es recull a la taula.

La taula compta, per cada valor de n, el nombre de vegades (sobre un total de 50 xarxes *overlay*) que cada algorisme ha donat millor resultats (en cas d'empat, no compten cap dels dos), indicat en la fila "best" (les comparacions les fem per separat, d'una banda entre els dos algorismes amb restriccions, i d'altra entre els dos algorismes sense restriccions). Com hem especificat com a criteri el temps de cadència t_{cad}, optcad és millor que optt0 un nombre elevat de vegades. Igualment, la taula inclou una fila amb valors de M. En aquest cas definim M com el mínim nombre de missatges que cal enviar per tal que l'algorisme optcad sigui millor que optt0. Com per definició optt0 prova de minimitzar el retard total t_0 de la transmissió d'un missatge (tal com es descriu a la secció 3.6) i optcad tracta de minimitzar el temps t_{cad} de cadència màxim, normalment el primer obtindrà millors retards t_0 en la transmissió d'un missatge, i el segon en la cadència de transmissió (millors t_{cad}). Si enviem M missatges el retard total de la transmissió és igual a:

$$t_{ret} = t_0 + (M-1)\cdot t_{cad} \qquad \textbf{(5.1)}$$

Com s'ha dit, t_0 és el retard en la transmissió d'un únic missatge i t_{cad} és el temps de cadència màxim de l'arbre multicast, és a dir, el temps que marca la separació d'enviament de dos missatges consecutius per evitar congestió i, per tant, el temps que marca la separació entre la recepció de dos missatges consecutius (per simplicitat, suposem que l'estat de la xarxa no canvia en el transcurs de la transmissió multicast). Aleshores, el retard total es calcula com el temps t_0 en què arriba el primer missatge, més el nombre de missatges restants M-1 multiplicat per la diferència de temps t_{cad} amb què arriben dos missatges consecutius.

Com optt0 obté normalment valors més petits de t_0 serà millor, en general, si enviem un sol missatge. Ara bé, com optcad obté valors més petits de t_{cad}, aleshores arribarà un moment en què si el nombre M de missatges és prou gran, la millora de t_{cad} compensarà en la fórmula 5.1 el pitjor valor de t_0 i aleshores optcad serà millor que optt0. Aquest valor de M és el que apareix a la taula (per als casos en què t_{cad} és menor per optcad que per optt0; en altre cas, si t_{cad} és menor per optt0, aleshores no té sentit calcular M perquè sempre serà millor optt0). En concret, es mostra el valor màxim, mínim i mitjà de M sobre els grafs *overlay* en què, com hem dit, optcad obté un millor t_{cad}. Veiem en general que el valor de M no és molt gran. En el pitjor dels casos val 2131,67 que es correspon (tenint en compte que els missatges, com hem dit, són paquets de 1500 bytes) amb una longitud de 3 Mbytes (paràmetres L de la taula). En promig, el valor més gran de M és d'uns 360 missatges, és a dir, corresponent a una longitud de 0,6 Mbytes, bastant habitual, per exemple, en transmissió de vídeo. En aquest cas, en la pràctica es podrien aplicar els dos algorismes (com hem vist, els dos s'executen molt ràpidament) i escollir-ne el millor. Si optt0 obté un t_{cad} més petit, aleshores el farem servir sempre. Si, contràriament, optcad obté el t_{cad} més petit (com passa en una majoria de casos quan no tenim restriccions: part de sota de la taula), aleshores el farem servir sempre i quan el nombre de paquets que hàgim d'enviar sigui més gran o igual a M, que és un valor que es pot calcular fàcilment a partir de la fórmula 5.1.

Finalment, si comparem els resultats de optcad amb els que s'obtenen amb l'algorisme "forced leaves" descrit a la secció 3.6 observem un fenomen curiós (aquests resultats es poden trobar a [18]) ja que en tots els casos aquest algorisme de forced leaves troba un t_{cad} idèntic al que aquí hem trobat amb optcad sense restriccions. Diem que això és curiós perquè optcad ha estat dissenyat per minimitzar t_{cad}, mentre que forced leaves s'ha concebut per minimitzar t_0. Això vol dir que en les nostres xarxes, com forced leaves té a més un t_0 més petit que optcad, podrem fer-lo servir sempre en comptes d'optcad. Això, que és cert, no vol dir que sempre sigui millor forced leaves, atès que les xarxes amb què hem treballat tenen una limitació particular.

Si analitzem node per node les xarxes *overlay* generades, veurem que totes contenen gairebé sempre un node amb un ample de banda molt petit que limita la cadència de tota la xarxa. Per exemple, la majoria de les xarxes tenen un peer amb ample de banda de 56 kbps, que dóna un temps de transmissió, per un paquet de 1500 bytes, d'aproximadament 214 ms. Això vol dir que en el millor dels casos tindrem un t_{cad} igual a 214 ms, ja que per definició el temps de cadència no pot ser més petit que el temps de transmissió d'un peer (com a

mínim un node ha d'enviar al peer més lent el missatge, i això suposa un temps de transmissió de 214 ms i per tant una cadència en un peer de com a mínim aquest temps). A la taula 5.3 es pot veure que optcad sense restriccions obté el mínim t_{cad} possible (214 ms) en tots els casos a partir de n=50. I per valors més petits de n es pot comprovar (estudiant xarxa per xarxa) que sempre obté el millor resultat, és a dir, que torna un t_{cad} igual al temps de transmissió més gran de tota la xarxa (temps forçat com hem vist pel peer amb menor ample de banda). Dit d'una altra forma, tenim xarxes molt descompensades amb peers que penalitzen molt el millor resultat possible. Això fa (i es pot veure al capítol de resultats de [18]) que altres algorismes com forced leaves puguin obtenir solucions iguals de bones que optcad, perquè hem treballat amb xarxes on aquesta possibilitat era senzilla. En qualsevol cas, és molt probable que en xarxes menys descompensades (com les xarxes on no n'hi ha gairebé peers amb un ample de banda molt petit) l'algorisme optt0 obtingui finalment millors resultats que tots els altres algorismes, punt que proposem per a una línia propera de treball.

6. CONCLUSIONS I TREBALL FUTUR

En aquest projecte, continuació de treballs anteriors [7,17,18], hem presentat una sèrie d'algorismes d'encaminament per la transmissió multicast de dades sobre xarxes *overlay*, és a dir, sobre xarxes definides al nivell d'aplicació. En particular, hem presentat un nou algorisme anomenat optcad que tracta de maximitzar la cadència de transmissió dels missatges o, dit d'una altra manera, de minimitzar el temps de cadència màxim de la xarxa (temps que, com hem dit, marca la separació entre la transmissió de dos missatges consecutius). S'han provat a més dos variants de l'algorisme: la primera amb restriccions per tal d'evitar que cap peer sigui més lent que la font. I la segona sense restriccions. Com s'ha vist, la segona possibilitat ha dibuixat arbres amb retards menors però amb un determinat nombre de peers més lents que la font. En aquest context, en tot cas, hem vist que podem evitar la congestió si limitem la cadència de la font al peer més lent de la xarxa, és a dir, si la separació entre dos missatges consecutius enviats per la font és com a mínim igual al temps de cadència del peer més lent de la xarxa (i.e. el temps durant el qual aquest node retransmet el missatge cap a d'altres peers).

L'impacte ambiental del projecte pot ser considerable si tenim en compte que pot escurçar el retard de transmissió dels missatges, la qual cosa vol dir economitzar ample de banda i per tant recursos, tant físics com energètics. Això, que en un xarxa aïllada pot tenir un impacte relatiu, podria prendre una magnitud considerable si el seu ús s'estengués a un conjunt gran de xarxes d'intercanvi d'informació.

Com a principal línia futura de treball es planteja, tal com s'ha explicat al final del capítol anterior i a curt termini, realitzar simulacions i comparacions sobre xarxes més realistes (o, com a mínim, sobre xarxes que no tinguin peers molt restrictius, és a dir, amb amples de banda molt petits) per tal de comprovar la teòrica superioritat de l'algorisme que hem anomenat "optcad sense restriccions". També es planteja la possibilitat d'estudiar els efectes de la connexió o desconnexió de peers sobre l'arbre multicast de transmissió. En aquest sentit, podem fer servir dues aproximacions: aplicar l'algorisme des de zero (és a dir, sobre tot el grup multicast) quan hi hagi un canvi; o bé provar de connectar els nous peers, o aquells que s'han quedat sense un peer proveïdor, a d'altres peers del grup. D'aquesta manera, l'algorisme podria permetre la connexió o desconnexió de peers amb l'objecte de preservar la connectivitat i minimitzar el retard total de la transmissió. Per últim, es planteja per més endavant desenvolupar un model matemàtic de l'algorisme (tal com s'ha fet a [18] amb l'algorisme MSM), introduir-ne petits ajustaments i fer finalment proves sobre xarxes reals.

BIBLIOGRAFIA

[1] AIMC, Asociación para la investigación de medios de comunicación, http://www.aimc.es/.

[2] M. H. Ammar, "Why Johnny can't multicast: lessons about the evolution of the internet", *NOSSDAV '03: Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*, New York, NY, USA: ACM Press, pp. 1-1, 2003.

[3] A. Bar-Noy, and S. Kipnis, "Designing Broadcasting Algorithms in the Postal Model for Message-Passing Systems", *ACM Symposium on Parallel Algorithms and Architectures,* pp. 13-22, 1992.

[4] A.-M. K. M. Castro, P. Druschel and A. Rowstron, "SCRIBE: A large-scale and decentralised application-level multicast infrastructure", *IEEE Journal on Selected Areas in Communication (JSAC)*, vol. 20, no. 8, October 2002.

[5] Y. Chawathe, S. McCanne, and E. A. Brewer, "RMX: Reliable multicast for heterogeneous networks", *INFOCOM* IEEE, pp. 795-804, 2000.

[6] S. E. Deering and D. R. Cheriton, "Multicast routing in datagram internetworks and extended LANs", *ACM Trans. Comput. Syst.*, vol. 8, no. 2, pp. 85-110, 1990.

[7] S. Domínguez, *Análisis de algoritmos multicast en redes overlay*, Treball Fi de Carrera, Escola Politècnica Superior de Castelldefels, 2006.

[8] E. W. Dijkstra, "A note on two problems in connection with graphs", *Numerische Mathematik,* vol. 1, pp. 269-271, 1959.

[9] Gibbons, *Algorithmic Graph Theory,* Cambridge University Press, Cambridge, 1985.

[10] GT-ITM, Modeling Topology of Internetworks, "http://www.cc.gatech.edu/computing/Networking/projects/gt-itm/".

[11] B. Huarte, *Optimización de rutas para el transporte urbano de mercancías,* Projecte Final de Carrera, ETSETB, Barcelona, 2004.

[12] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole, Jr., "Overcast: Reliable multicasting with an overlay network", in *USENIX Symposium on Operating Systems Design and Implementation*, pp. 197-212, 2000.

[13] Jeff McAffer, Jean-Michel Lemieux, *Eclipse Rich Client Platform: Designing, Coding and Packaging Java Applications*, Addison-Wesley Professional, 2005. Web site: "http://eclipsercp.org/".

[14] NED Network EDitor, “http://www.cs.nyu.edu/pdsg/projects/partitionable-services/ned/ned.htm”.

[15] The Network Simulator NS-2, “http://www.isi.edu/nsnam/ns”.

[16] J. Ozón, *Contribución al coloreado de grafos y las redes pequeño mundo*, Tesis Doctoral, Universitat Politècnica de Catalunya, 2001.

[17] J. Pratsevall, *Anàlisi d'algorismes de multicast en xarxes overlay*, Treball Fi de Carrera, Escola Politècnica Superior de Castelldefels, 2005.

[18] J. Pratsevall, *Application-Layer Multicast Algorithms for Bounded Delay Transmissions,* Master Thesis, Escola Politècnica Superior de Castelldefels, 2008.

[19] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, “A scalable content addressable network”, in *ACM SIGCOMM*, 2001.

[20] J. H. Saltzer, D. P. Reed and D. D. Clarck, “End-to-end arguments in system design”, *ACM Trans. Comput. Syst.*, vol. 2, no. 4, pp. 277-288, 1984.

[21] R. J. Wilson, *Introducción a la Teoría de Grafos,* Alianza Universidad, Alianza Editorial, Madrid, 1972.

[22] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, “How to Model an Internetwork” in *IEEE Infocom*, pp. 594-602, IEEE, San Francisco, 1996.

[23] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz, “Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination”, in *NOSSDAV*, June 2001.